Guidelines for Practical Algorithmic Design for Industrial Designers

by

Andrew Davis Edge

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Industrial Design

Auburn, Alabama
May 5, 2019

Keywords: Industrial Design, Algorithm, Computer-Aided Design, Generative Design, Digital Fabrication, Visual Programming

Abstract

Algorithmic design is the use of a particular set of instructions to define the building process for a product or geometry. New computer-aided technologies have emerged which allow for a closer connection between the designer's concept and its creation. Using algorithmic design processes to assist in computer-aided design can be very powerful, but unwieldy if used improperly or in the wrong context. This study researches the properties and benefits of algorithmic design and algorithmic modelers in order to develop a set of guidelines to assist the industrial designer through the algorithmic design process, particularly in deciding when and in what context algorithmic design is an appropriate avenue for the designer to take. This research also provides a set of commonly used algorithmic concepts and techniques to aid in the algorithmic modeling process. Additionally, the guidelines and techniques are demonstrated through an industrial design project wherein a concept cell phone case is designed, evaluated, and brought through the algorithmic modeling process.

Acknowledgments

To Mom and Dad for supporting me as I went off in this new direction,

To Alex, Jess, & Aaron for having adventures with me when work was over,

To Alecia Jo, for being my biggest fan and best friend,

To Waz, Chandler, Taylor, Adria, Zak, Jennifer, and Clay for welcoming me,

To all of the Auburn ID professors for their mentorship,

And to my other friends old and new for listening,

Thank you all.

Table of Contents

Definition of Key Terms

3D Model: Mathematical representation of a geometry within a 3D modeling software

3D Printing: The manufacturing of solid objects by the deposition of layers of material (such as plastic) in accordance with specifications that are stored and displayed in electronic form as a digital model ("Dictionary and Thesaurus | Merriam-Webster", 2018).

Abstraction: The process of removing physical, spatial, or temporal details (Colbourn, 2007)

Algorithm: A step-by-step procedure for solving a problem or accomplishing some end ("Dictionary and Thesaurus | Merriam-Webster", 2018)

Algorithmic Design: The process of using algorithms to define systems or geometric forms

Computer Aided Design (CAD): The use of computers (or workstations) to aid in the creation, modification, analysis, or optimization of a design (Narayan, 2013)

Digital Fabrication: Process by which machining tools and other digitally-controlled manufacturing methods are used, in order to replicate the designs created in a CAD software

Generative Design: The process of defining limits, forces, and other constraints on a system and producing an output that fits within the parameters

Geometry: A branch of mathematics that deals with the measurement, properties, and relationships of points, lines, angles, surfaces, and solids ("Dictionary and Thesaurus | Merriam-Webster", 2018)

Industrial Design: Design concerned with the appearance of three-dimensional machine-made products and the study of the principles of such design ("Dictionary and Thesaurus | Merriam-Webster", 2018)

Iteration: A procedure in which repetition of a sequence of operations yields results successively closer to a desired result ("Dictionary and Thesaurus | Merriam-Webster", 2018)

Parameter: Any of a set of physical properties whose values determine the characteristics or behavior of something ("Dictionary and Thesaurus | Merriam-Webster", 2018)

Visual Programming: The use of a graphical interface to form programs and functions rather than a text-based interface

Working Drawing: A scale drawing of an object to be made or structure to be built intended for direct use by the workman ("Dictionary and Thesaurus | Merriam-Webster", 2018)

Chapter 1: Introduction

Problem Statement

An algorithm is an unambiguous set of properly defined instructions (Tedeschi, 2016). This is a simple statement, but these sets of instructions can be used to define near-infinitely complex systems. Algorithmic modeling allows for the utilization of these algorithms for the formation of geometry within computer-aided-design (CAD) programs. The benefits of learning these strategies and programs can be invaluable to a designer, granting methods to create geometries and utilize information sets which would be difficult, if not impossible, to imitate through traditional CAD software packages. However, algorithmic modelers can be difficult to operate, and require a greater time commitment than traditional CAD packages. At its core, the algorithmic design process requires a completely different method of thinking and planning in order to create desired geometry. Moreover, the study and use of algorithmic design has been largely in the realm of architecture, as seen in textbooks, online resources, and even college curriculums. In a modern design environment, industrial designers and traditional CAD modeling have a close relationship, but algorithmic modeling is a fringe activity. This document will not be a comprehensive algorithmic modeling guide, but by developing guidelines for the use of algorithmic modelers for industrial designers, opportunities can be created for industrial designers to take advantage of the benefits of the algorithmic design process and demystify its operation and implementation.

Need for Study

The rise of digital fabrication has provided an opportunity for nearly anyone to manufacture or prototype parts and products, with the main requirement being that they have a properly defined "digital" model of their part or system. The translation from digital design model to digital fabrication is becoming faster and more fluid every day, therefore, so too must the connection between design concept and digital design model become more fluid. Often, with the introduction of new design technologies or manufacturing techniques, the possibilities for production in new geometries and forms are expanded. With the introduction of algorithmic modelers, geometries with increased complexity can now be created and controlled with greater ease, shortening the gap between design concept and digital model.

Algorithmic modelers provide increased flexibility for parametric design processes. The methods by which algorithmic designs are built, with defined inputs producing predictable outputs, allow for easily variable inputs to create near infinitely variable outputs. This rise in parametric power doesn't come without cost though, as there is more set-up work earlier in the design process for algorithmic systems. The benefits of algorithmic design are many, but the ability to utilize it effectively is difficult to master. This study will focus on creating a set of guidelines for industrial designers for the use of algorithmic modelers, answering what algorithmic design is, how to use it effectively in product design, and importantly, under what circumstances algorithmic design should be used. This document will also outline common algorithmic modeling terms, features, and algorithmic methods which can serve as a starting point for industrial designers to form their own algorithmic system designs upon.

Objectives

The areas of research listed below will be used to develop guidelines for the practical use of algorithmic design within industrial design. This thesis will also provide examples of use for different strategies and methods in algorithmic design.

Objectives:

- Research algorithmic design theory
- Research case studies for common algorithmic design processes
- Research methods for implementing algorithmic design
- Research common uses for algorithmic design and translate into industrial design use cases
- Develop guidelines for industrial designers to know when and how to use algorithmic design in their models
- Perform an industrial design project process that demonstrates the use of the guidelines.

Assumptions

The information and guidelines conveyed in this study are a product of the current technological climate. The methods and strategies for algorithmic modeling processes are presented in their current programs and their details are subject to change as technology progresses. Additionally, the information presented here uses digital fabrication, such as 3D printing, as an eventuality for the models since there is a basis of using complicated algorithmic designs in digital form to control CNC systems. However, the complex geometry that can be created within algorithmic modelers may not be producible through certain manufacturing or digital manufacturing processes. It is assumed that the designers can recognize the limits and constraints of the manufacturing processes they plan to use. This document lists several key considerations for the decision to use or not use algorithmic design. These considerations can help industrial designers, but they must be able to evaluate their own skills and the design situation at hand to determine their best course of action.

Scope and Limitations

Considering the immense space that algorithmic design could encompass, this study will be limited to algorithmic design and formulation of geometric shapes to be used within computer-aided-design programs. This is not to say that algorithmic design is only a tool for 3D modeling; it is an abstract way of thinking that can be applied to any number of situations. It should also be understood that although this document will include many examples of works and methods related to algorithmic design, there are no real limits to what can be created; therefore, many strategies and specific methods might not appear here. Industrial design is a large, varied field including UX, systems design, packaging, branding, etc. and there may be ways in which the concepts behind algorithmic modeling can help in these areas, but references to the industrial design process in this document will be in regards to form development and physical geometry concerns only.

Anticipated Outcomes

The end result of this study should produce a set of guidelines for use by industrial designers, which will help ease their introduction into algorithmic modeling and provide them with insights into how, why, and when to use algorithmic modeling. This study will describe various methods and strategies for developing algorithmic models for the formation of geometries used with industrial design and will attempt to translate common use cases in algorithmic modeling into possible design application spaces within industrial and product design. Providing industrial designers with guidelines for the use of algorithmic modeling will allow them to take advantage of the many benefits that come with the algorithmic design process, but also, importantly, will allow them to know which situations will and will not be good candidates for algorithmic design work. This document will also provide an extensive industrial design example project to demonstrate the use of the guidelines and an algorithmic modeler.

.

Chapter 2: Literature Review

Working Drawings

The translation of designs from the minds of the creators to the creations themselves often requires a design planning process as a translation medium. This process allows for organization, prediction, and alteration of ideas before any physical formation begins, and before the large construction costs. The methods by which these design plans and working drawings are formulated has varied throughout history and the nature of these methods can inform the nature of the designs. In the opening lines of *AAD, Algorithms-Aided Design: Parametric Strategies Using Grasshopper*, Tedeschi (2014) states:

> Architects have always drawn before building, an act that differentiates architecture from the mere construction. Drawings have been the architect's medium to organize ideas, resources, space, etc. and represent the architects' faculty to predict design outcomes. As methods of representation have evolved, new styles have emerged. [...] Tools such as perspective in the Renaissance and projective geometry in Modernism have marked leaps forward in design. However, these tools have been dependent on a stable set of instruments for centuries: paper, drawing utensils, ruler and compass. In this model, each creative act is translated into a geometric alphabet by gestures, which establish a direct link between the idea and the sign (p. 15).

However, the origins of design planning and working drawings begin further back than even pen and paper. Some of the earliest known, but surviving, working drawings were found in 1979 on the walls of the Temple of Apollo at Didyma, which began construction ~334 BCE. Haselberger (1985) details how these drawings provide some insight into early planning and construction methods, where the plan and the material were one and the same. The Temple of Apollo working drawings depict plans for columns, etched into the walls and marbles where they would be constructed. The reason these drawing survive is due to the fact that this portion of the temple was never finished and the etched plans were never brought to completion. Even though the designers of these columns created detailed drawings, there was room for the designer to insert their own flair and ideas into the final form. Shown in Figure 1, the original blueprint lines, shown in red, were altered during the carving process, adding cutouts and detailed changes.

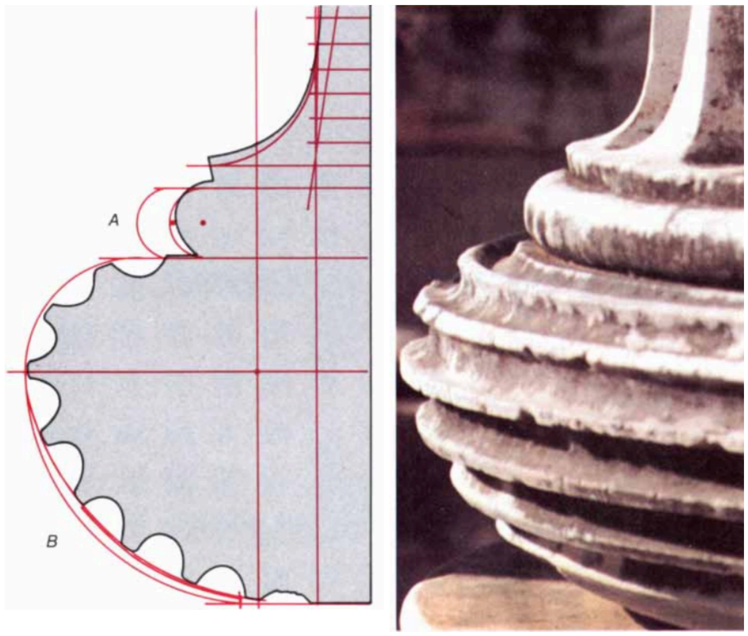


Figure 1: Left: Representation of the Didyma blueprint carvings shown in red. Right: Finished column at the Temple of Apollo. © Scientific American 1985, Haselburger.

The relationship between the designer and the construction medium seen at the Temple of Apollo was direct, a 2D representation on the 3D building material. This relationship can still be found today in small construction projects and woodshops where designers will draw their next cuts onto a material, but it is not suitable for the increased complexity and scale of modern manufacturing.

With the advent of paper, clear examples are found of its use in working drawings and architectural plans such as in Figure 2, the *Plan A1 of the Facade of the Strasbourg Cathedral*. These plans were painstakingly produced by hand when needed. Early construction plans found in the Middle Ages, such as *Plan A1*, are orthographic representations of the final work, or in other words, 2D representations on a 2D medium (Holcomb & Bessette, 2009). The relationship between designer and construction medium has now shifted away from the material itself. Plans are developed with increased complexity and allowing for better transferring of design information.

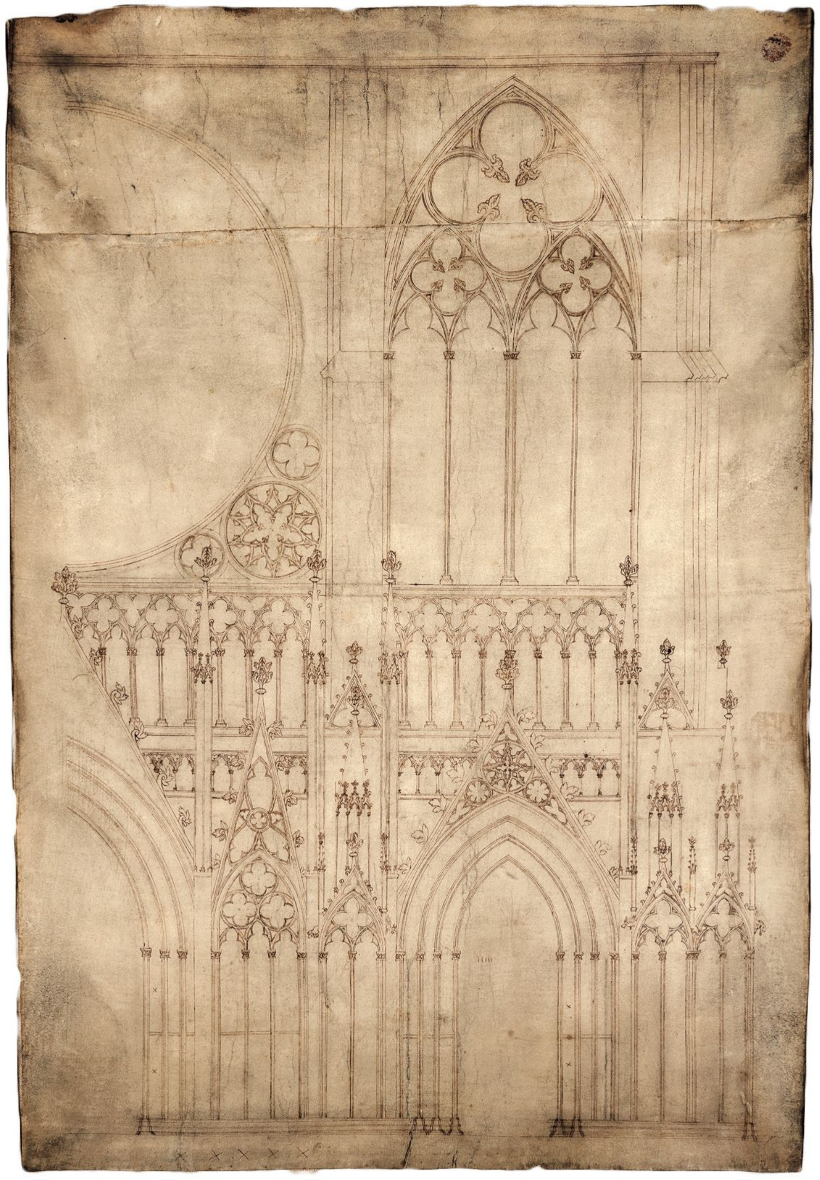


Figure 2: Plan A1 of the Facade of the Strasbourg Cathedral, (Holcomb & Bessette 2009)

The Renaissance introduced perspective drawings into the design process. Filippo Brunelleschi is often regarded as the father of perspective drawings, as he helped codify and document rules and strategies for their creation (Argan & Robb, 1946). His techniques can also be seen in works such as his elevation of Santo Spirito, a perspective drawing of the church interior. Brunelleschi was able to show his patrons a representation of what the final construction of Santo Spirito would look like, based only from the 2D floor plans currently available. The standard has now moved into 3D representations on a 2D medium, allowing for more information to be conveyed by the designer, without expanding the medium (Argan & Robb, 1946)

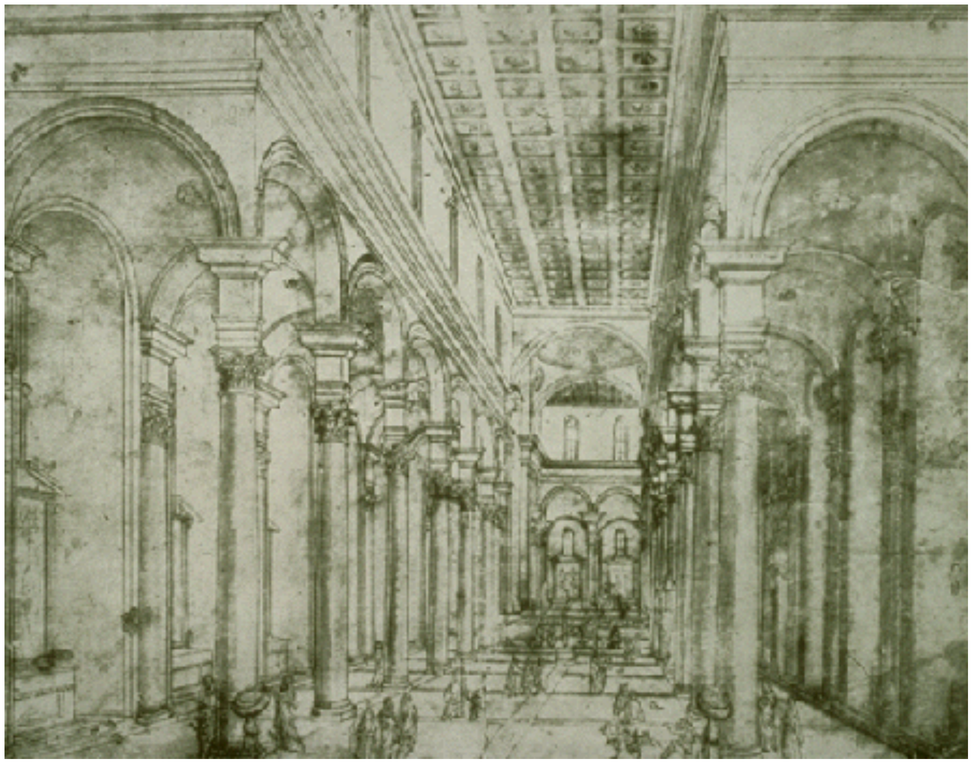


Figure 3: Perspective drawing for the Church of San Spirito in Florence (as cited in Argan & Robb, 1946)

Similarly, Leonardo Da Vinci, another Renaissance artist and engineer, was a master of perspective drawing but also was one of the first to successfully represent designs as technical drawings similar to what a modern designer or engineer would develop. Da Vinci's drawings often included exploded views, detail views, section views, and copious notes. The detail and forward-looking thought process behind the drawings allowed Da Vinci to evaluate his concepts, while they still just existed on paper (Isaacson, 2018). This analysis is shown in his drawings of a concept perpetual-motion machine, shown in Figure 4. After sketching out and evaluating his design, he deemed it as an impossible venture.

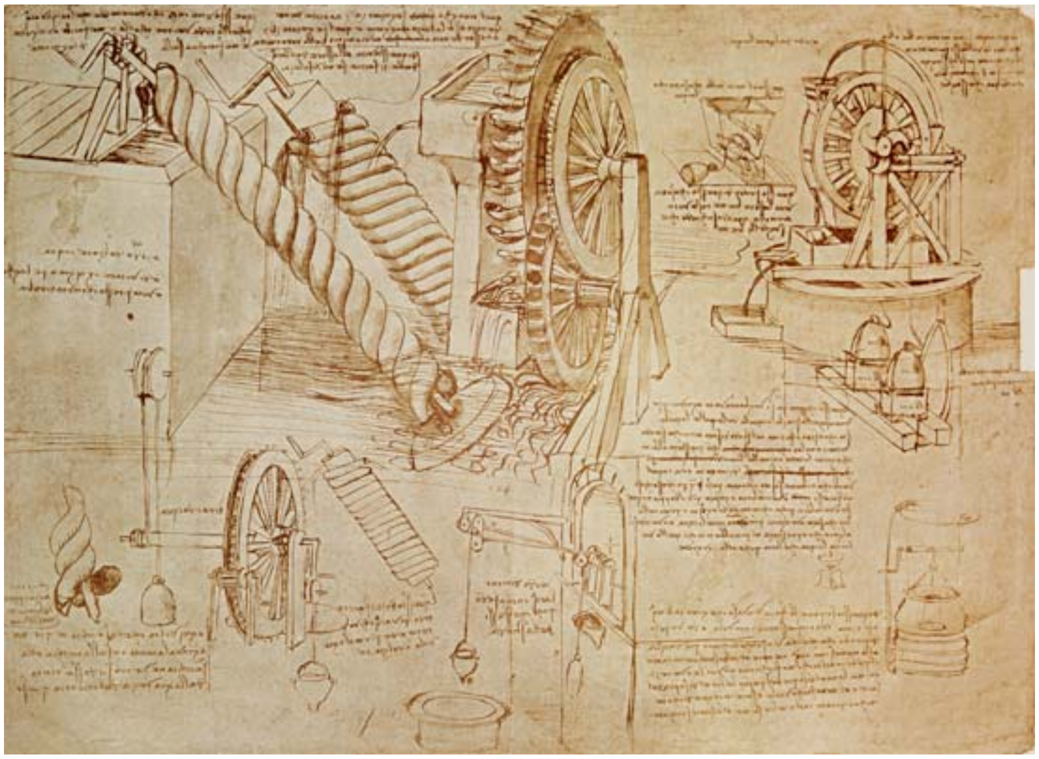


Figure 4: Da Vinci's water screw perpetual motion machine. (as cited in Isaacson, 2018)

As the Industrial Revolution begins, it's shown that the beginnings of mass production have created the need for increased precision and repeatability in parts and in designs. This need for precision requires increased sophistication in the part drawings and technical drawings of the time, as well. French mathematician, Gaspard Monge, was the first to develop descriptive geometry, a set of standardized rules and methods for representing 3D geometries in a 2D space for the purposes of production (Carlbom &

Paciorek, 1978). Even today the "Front, Right, Top, Perspective" layout popularized by Monge is used in technical/engineering drawings to describe parts. Monge also described various methods of projecting and translating geometric views, shown in his *Géométrie Descriptive*.

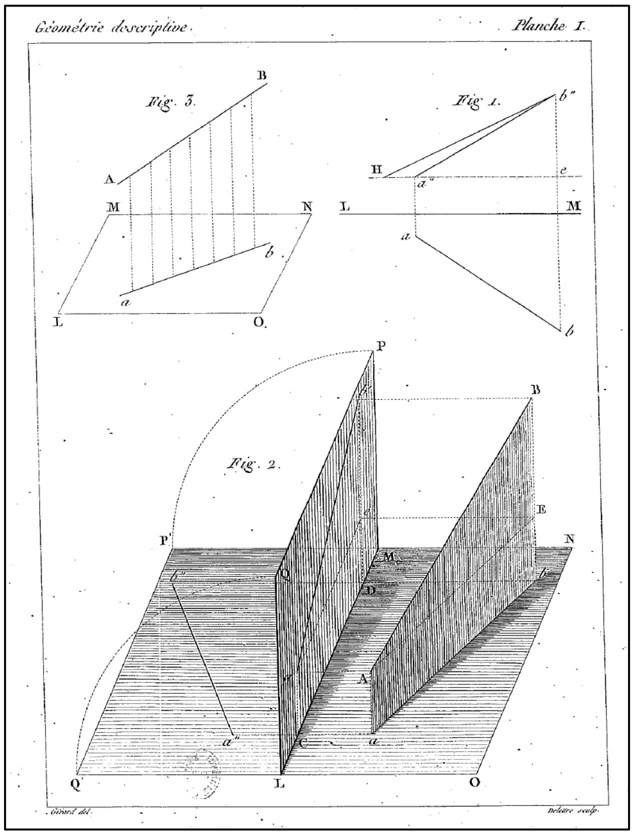


Figure 5: Projection illustrations in *Géométrie Descriptive* (as cited in Monge, 1798)

From the Industrial Revolution to the 1960's, the methods by which engineers and designers produced working drawings remained largely unchanged. As projects and designs grew larger and larger, however, the physical act of creating the working drawings fell to the now well-established profession of draftsmen. Draftsmen are skilled workers who create and document designs using tools not altogether different from those found even in the Middle Ages: straight-edges, curve tools, compasses, etc. As projects grew increasingly complicated, more and more draftsmen were required to tackle them, each working on a separate portion of the project until whole floors of company buildings were devoted as draftsmen workspaces. This was the norm until the invention of the computer in the 1960's and subsequent CAD programs made draftsmen rooms, such the one shown in Figure 6, obsolete. Working drawings are still used in documentation and construction, but now they are created on the computer. Even with modern advances, the standard design process is using the computer to translate the design from the creator's mind, the concept, to a 3D representation, the model, which is then translated to a series of 2D representations, the working drawings, which are then used to create the object. This process still relies on several translations and a series of steps, which separate the designer from the end result.

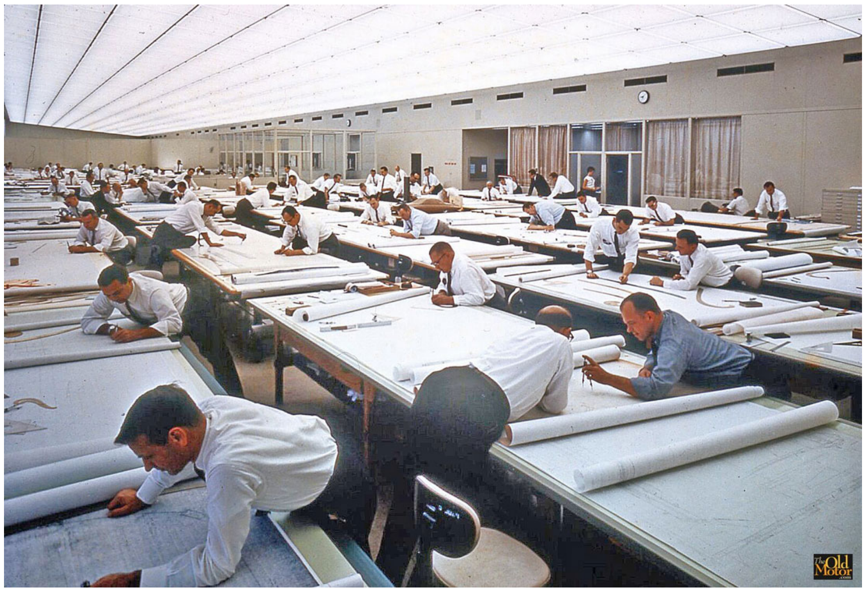


Figure 6: General Motors Styling Section drafting room, 1960's ("The Old Motor", 2015)

Computer-Aided Design

In *Computer Aided Design and Manufacturing,* (Narayan, 2013) defines computer aided design as:

> The use of computer systems to assist in the creation, modification, analysis, or optimization of design. [...] CAD helps the designer to visualize the product and its component sub-assemblies and parts. This reduces the time required to synthesize, analyze and document the design. This productivity improvement results not only into lower design cost but also into shorter design project completion times. (p. 3)

These programs are used in many different fields including animation, medicine, and computer science, but are an indispensable part of the modern design and engineering process. This study will focus mainly on 3D solid modeling as opposed to organic modeling and, at times, more specifically on parametric modeling. Three dimensional modelers allow us to develop geometry, create part drawings, perform various analysis on parts, send those models to be manufactured by digital manufacturing processes, and much more. CAD programs so greatly simplified the design process they eliminated the need for massive amounts of draftsmen to be employed by companies. The beginning of the digital era in the early 1960's, was marked by the advent of the computer and the explorations into applications for this new device.

In 1963 MIT PhD candidate Ivan Sutherland showcased his thesis, Sketchpad. Sketchpad was the first of many things, including the first interactive graphics program, the first non-procedural programming language, and the first object-oriented software system. Put simply though, Sketchpad was a computer drawing system, which allowed the user to create lines and shapes, define relationships, copy and paste sketch entities, and even allowed for some rudimentary 3D visualization of sketches. Although not his full intention on the outset, when Sutherland developed his Sketchpad system, he became the inventor of the field of computer-aided-design.

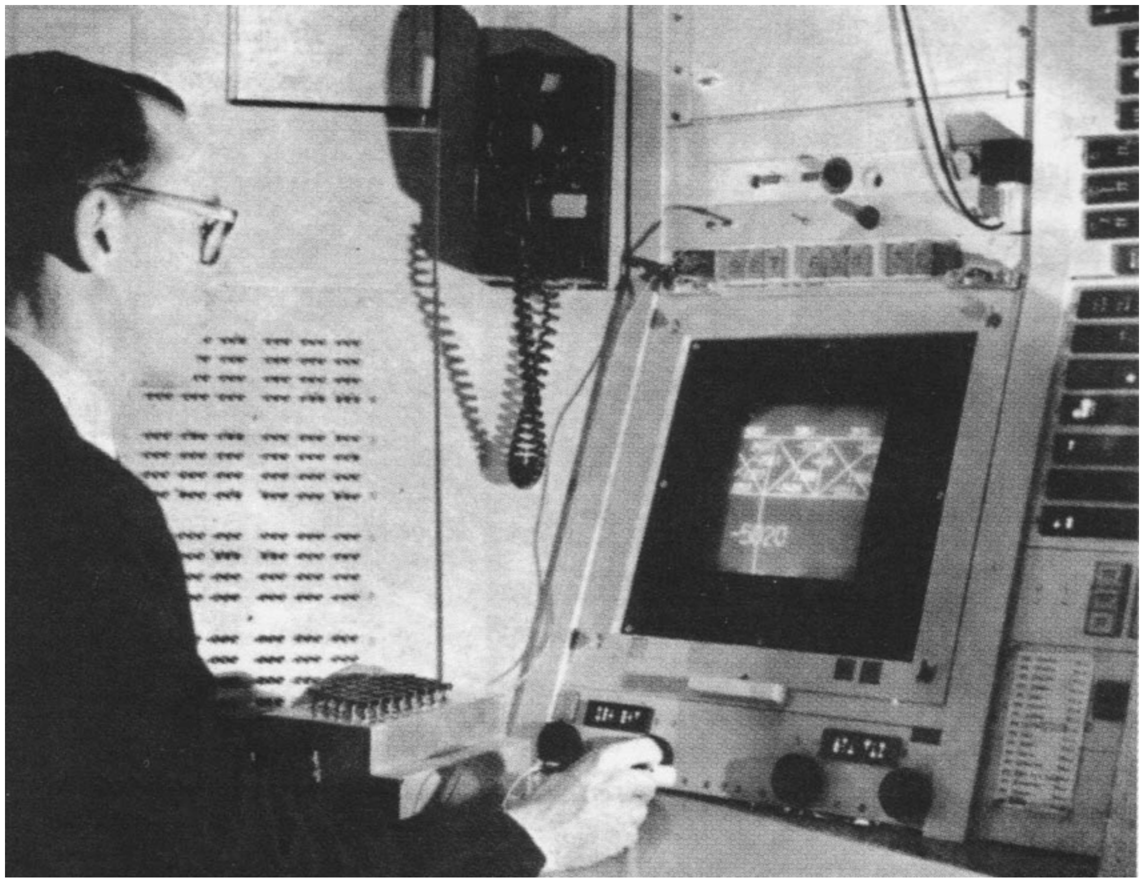


Figure 7: Computer scientist working with the Sketchpad program ("Computer History Museum", 2019)

In 1968, soon after Sutherland's breakthrough, Pierre Bezier, a French mathematician, developed UNISURF, the first wireframe modeling CAD system. A wireframe modeler operates by defining the boundaries of a form with curves. The Bezier curve, popularized but not invented by Bezier, is a mathematically defined curve that wireframe modelers utilize to approximate a 3D form. This development led to the modeling of curved surfaces within a digital space.

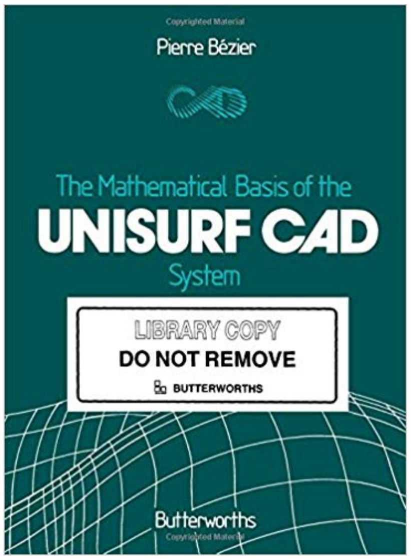
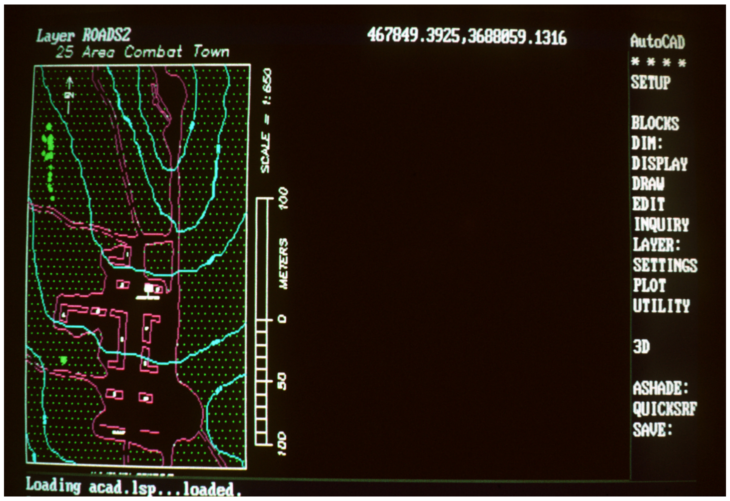


Figure 8: *The Mathematical Basis of the Unisurf CAD System (Bezier, 1986)*, with graphics depicting surface representations made from Bezier curves.

Throughout the 1970s, many CAD programs were developed in-house at large automotive and aerospace firms: General Motors, Ford, Lockheed, Northrop, etc. Each company wanting to gain an advantage over others with more and more advanced systems. However, this segmentation of the CAD systems meant that there were no standards or means of communication between the systems. Large clients of multiple firms, such as the United States Air Force, pushed for more standardization between the different CAD systems. This led to the formation of the IGES (Initial Graphics Exchange Specification), a digital exchange format for CAD systems, in 1980. This allowed competing CAD programs to communicate with each other, narrowing the divide among the programs (“Cadazz”, 2004).

In 1981, IBM shipped the first personal computer (PC) and signaled a new need for CAD systems to operate on these workstations. Officially founded in 1982, Autodesk released the first CAD software for PC, AutoCAD, which became the primary CAD system worldwide, by the late 80’s. AutoCAD however, was still primarily a 2D program, mostly used for technical drawing representations of parts and floor plans.



Loading acad.lsp...loaded.

Figure 9: Original AutoCAD interface

The first 3D CAD software available to the public was PTC’s (Parametric Technology Corporation) Pro / Engineer, released in 1987. Pro / Engineer was the first 3D CAD system to be based entirely on solid models, and history based features and constraints, otherwise known as parametric modeling. 3D modelers allow for models to be fully defined in 3D space, so information can be ascertained from any possible angle without having to redraw the model by projecting the model onto a 2D surface. In other words, if the model is designed fully in 3D, all 2D representations of the model are trivial to develop. The other main benefit that Pro / Engineer brought was an interface which was much more usable than its predecessors, now including dropdown menus, icons, input boxes, etc. (“Cadazz”, 2004).

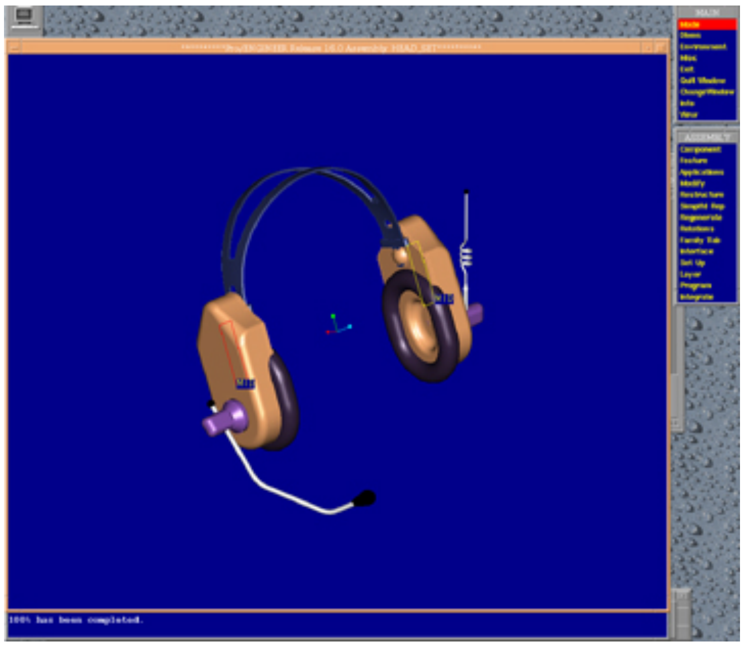


Figure 10: Original Pro/Engineer interface

Over the next 20 years after the release of Pro / Engineer, the story of CAD is one of incremental changes. In the modern technological environment, CAD programs are incredibly powerful and readily available to anyone with a computer, not just employees of large manufacturers. File standardization above and beyond the original IGES is now the norm, allowing all the major CAD programs to communicate. AutoCAD, Solidworks, CATIA, Autodesk Inventor, PTC Creo, and Rhino 3D, are just a few among today’s most popular programs, all descendants of the original programs of the 1960’s.

Parametric and Direct Modeling

Within the 3D solid-modeling field, there are two main categories of modeling, parametric and direct modeling. These categories are notoriously difficult to define, and the lines between the categories can be blurred in certain programs. Generally, parametric modeling allows the user to define features with specific parameters that form and change the model, and these features are captured within a design history. If built correctly, these features can be adjusted and updated to further define form of the model, and can be retroactively changed. The parametric design environment usually allows for parent-child relationships to be made between models, so changes in one model file will be reflected in other files that call upon the original file. This relationship allows families of models to be created where design changes can be made at any point. The mathematical definitions behind the features of a parametric model also allow for easy integration into manufacturing tools and digital fabrication methods (Narayan, 2013)

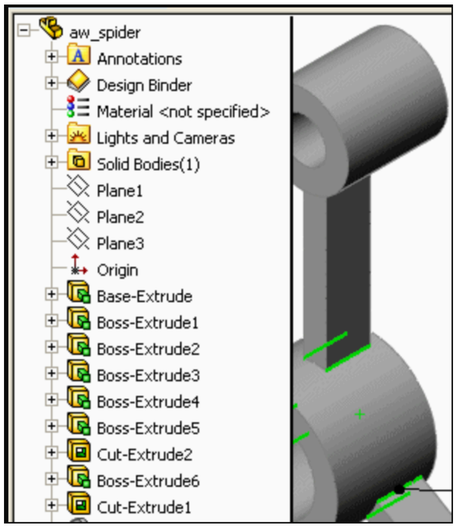


Figure 11: Feature tree from Solidworks

Alternatively, direct modeling programs create and adjust geometry through direct means, by moving selected faces, control points, and parts. Usually, direct modelers do not capture any design history during use so, once a change is made, it is not possible to adjust that change later in time. Direct modeling allows for a straightforward "sculpting" strategy, but can lead to problems in fields where changes may be needed in the future.

There are some programs such as Fusion360, shown in Figure 12, which are hybrid systems between parametric and direct modeling, where the designer can utilize both methods in the same design file.

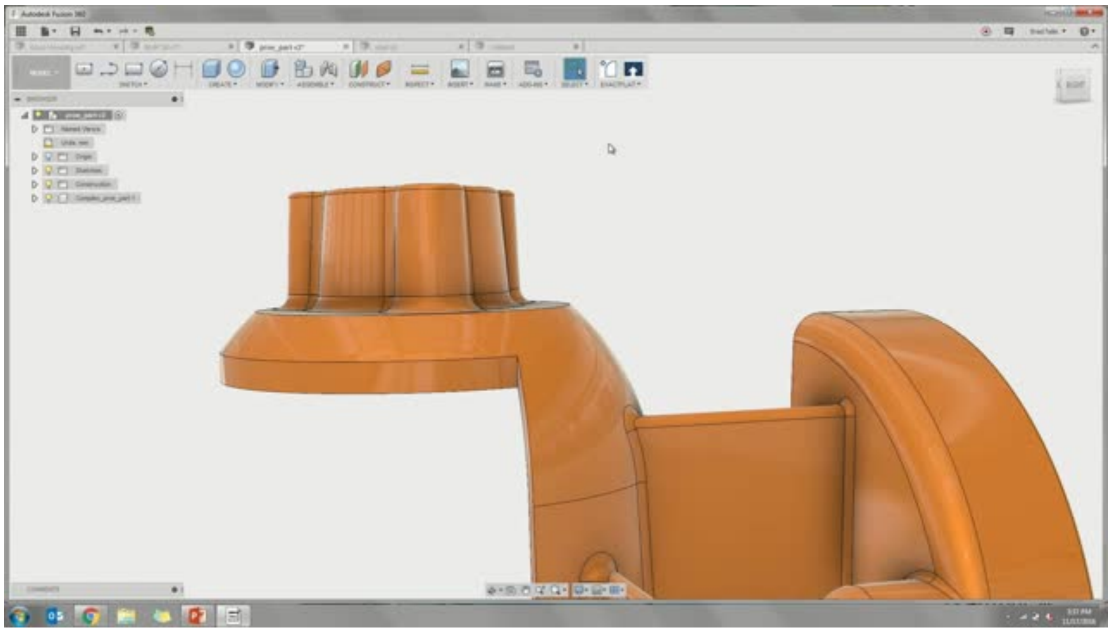


Figure 12: Fusion 360 modeling interface

Algorithmic Design

Algorithmic design is not just constrained to the digital realm In *Expressive Form, Terzidis (2003) describes an algorithm as:*

> "…a computational procedure for addressing a problem in a finite number of steps. It involves deduction, induction, abstraction, generalization, and structured logic. It is the systematic extraction of logical principles and the development of a generic solution plan." (p. 94)

Any set of instructions or steps which have inputs and a predictable output can be thought of as an algorithm. Baking a cake, tying shoelaces, and doing laundry, all can be broken down into individual actions and, with enough detail, can be described in such a way that a "Baking Cake Algorithm" or "Laundry Algorithm" can be produced.

Algorithmic modeling, while it does have its own set of connotations and uses, is, at its core, a more in-depth form of parametric modeling. It allows for clearly defined steps to be put down which will eventually create geometries and designs. Algorithmic designs can be either Inventions or Discoveries (Terzidis 2003). Inventions occur when the designer defines actions for the algorithm to take so that a specific outcome will be reached. A Discovery-based algorithm occurs when the designer provides bounds and directions for the algorithm to move in, but the final outcome is not fully predictable. This is also known as generative design, but the focus of the guidelines of this document will be upon the Inventive side of algorithms. It will be shown later though, that even in Inventive systems, there can be enough variability in the parameters created to have somewhat unpredictable outputs.

The program that is being used to model the algorithm often limits the depth of parametric control needed for algorithmic design. Many 3D modeling programs do offer tools that can help create fully parametric models, but the ability to create changeable variables and control relationships between different elements is much more easily done in a visual programming language or algorithmic modeling program. Sakamoto (2008) explains:

> "The instrumentation of parametric setups into architectural practice is starting to shift the role of the architect in the design processes: from the design of specific shapes to the determination of those geometrical / algorithmic relationships describing the project and its components. The design shifts from drawing surfaces to setting up rules of interdependency – genotypes – leading to potential differentiation – phenotypes." (p. 118)

Algorithmic modeling is used in a wide range of applications but most notably within the fields of architecture, engineering and digital fabrication. In architecture it allows for many different factors to be taken into account at once, lighting, walking space, seating, heat, ventilation, noise and can help create solutions that balance all of these. In engineering, algorithmic modeling is being used to help create optimized structures, which can withstand higher stresses with less material. In fabrication, algorithmic modeling is helping create geometries that would otherwise be impossible to model using digital manufacturing techniques such as 3D printing and advanced CNC methods to fabricate these forms. Figure 13 shows an example of algorithmically designed architecture where the structure of walls and ceiling of the Serpentine Pavilion were determined using a tessellation algorithm. The structure takes into account views to the street, surrounding building heights, and sun-paths (Agkathidis, 2016).

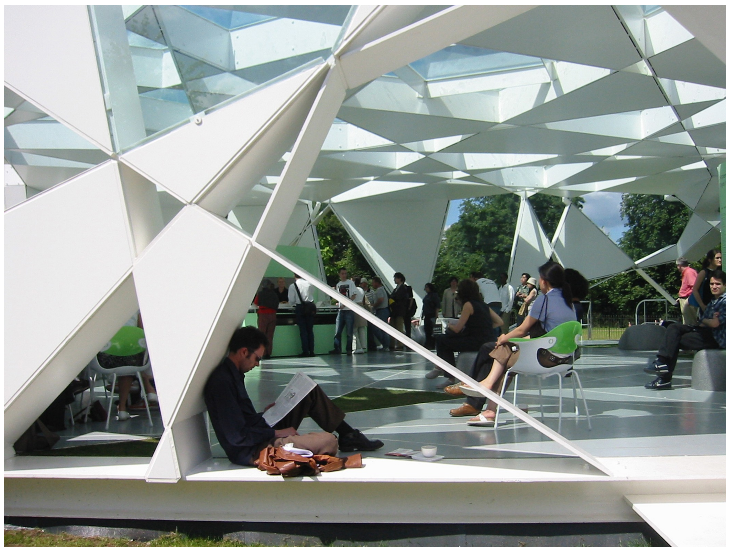
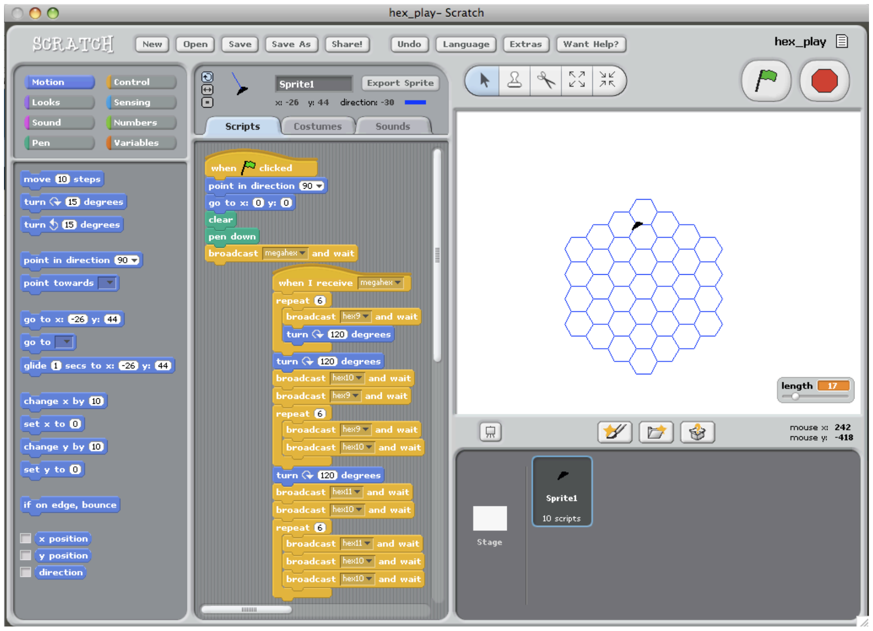


Figure 13: Serpentine Gallery Pavilion 2003

It is important to note that there are two outputs to the algorithmic modeling process: the model, and the method. The algorithmic modeling process will produce a 3D model of the design to be used for fabrication, rendering, etc., but the algorithm itself is valuable as well. The algorithm can be reused in other contexts by different people. It is easily sharable and can be adjusted for different uses.

Visual Programing

Algorithmic modeling is often completed through the use of visual programming languages (VPL). A visual programming language is one that allows the user to code systems and, in this case, define geometry, through a graphical interface instead of through lines of code. The step-by-step nature of algorithmic modeling lends itself to an object-based coding environment, and the graphical interface of a visual programming language allows for people with little coding experience to work more effectively. Typically, a visual programming language has a workspace where the user can place nodes and connections. Nodes can either perform a certain function or be given a certain value. Nodes that perform functions usually have a place to connect inputs and outputs. By connecting values to the inputs of functions, which then output values, which can then be input into other functions and so on and so on, complex systems can be created without ever having to “write” a line of code. Visual programming languages often act as introductory systems into the world of coding for young coders and non-coding professionals. It gets rid of the need to learn the specific syntax and punctuation requirements that are found in most text-based coding systems (Levy, 2017).



```
when [flag] clicked
point in direction 90
go to x: 0 y: 0
clear
pen down
broadcast megahex and wait

when I receive megahex
repeat 6
    broadcast hex9 and wait
    turn 120 degrees
turn 120 degrees
broadcast hex10 and wait
broadcast hex9 and wait
repeat 6
    broadcast hex9 and wait
    broadcast hex10 and wait
turn 120 degrees
broadcast hex11 and wait
broadcast hex10 and wait
repeat 6
    broadcast hex11 and wait
    broadcast hex10 and wait
    broadcast hex10 and wait
```

Figure 14: Scratch, a visual programming interface

The two algorithmic modeling systems this document will cover are Grasshopper 3D and Autodesk Dynamo. These programs are VPL systems where the user can place nodes that perform functions that create geometry, perform actions upon that geometry, or can provide information and data about geometry. By stringing together nodes, complicated data transformations can create complicated geometries, which are not possible in other programs. There are, of course, more VPL programs used for the purpose of creating 3D geometry, but Grasshopper and Dynamo are the most widely used and easily available programs.

Grasshopper

Grasshopper 3D, or simply Grasshopper, is a VPL and algorithmic modeling environment created by David Rutten at Robert McNeel & Associates. Originally, Grasshopper was a downloadable plug-in command, which ran within Rhino 3D software, but now has been integrated into the standard Rhino toolset for Rhino 6.0. In its current form, the Grasshopper environment appears as separate windows where the nodes and connections can be placed which define geometry, which are visually represented within the Rhino environment in the original window. The user can build geometry in Rhino, which can then be used as inputs for algorithmic processes in Grasshopper, which the user can then “Bake” back into the Rhino environment to be further manipulated using traditional processes. There is also a large community of users and creators who develop Grasshopper tools for download which are easily installable into Rhino (Grasshopper, 2018).

Autodesk Dynamo

Dynamo Studio is an algorithmic program developed by Autodesk. It is also a VPL and environment which operates in a very similar fashion to Grasshopper 3D. Details aside, one of the main differences is that Dynamo can operate as a standalone product but typically is used in conjunction with Autodesk Revit, and architectural modeling (BIM) software. In Dynamo, the VPL workspace is layered on top of the geometry visualizations instead of in two windows, as seen in Grasshopper (Dynamo Primer, 2018).

Generative Design

Generative design is the use of algorithmic processes and modelers to create forms and solutions that are not explicitly dictated by the algorithm writer. A generative design process has the user defining goals, constraints, and other general rules for a system to follow, and the system will attempt to abide by them. Generative design is used in generative form-finding, optimization, and processes where the ultimate design is to be discovered and not dictated. *Generative Design* (Agkathidis, 2016) describes four areas of generative cases: design processes driven by nature, geometry, context, and performance. Design driven by nature is an attempt to mimic the forms and patterns found in the natural world. Design driven by geometry is created by following mathematical principals and geometrical proportions to develop form. Design driven by context is based upon the particular situation at hand: the historical, cultural, geographical, and material particularities of the situation can drive the design of a project. Design driven by performance attempts to maximize or minimize a certain aspect of a design, be it weight, strength, openness, temperature etc.

There are multiple programs available for use to leverage algorithmic modeling for generative design but one of the most easily available ones is the Kangaroo Physics program created by Daniel Piker, which is now included with Rhino 6.

Digital Fabrication & Mass Customization

Digital fabrication is the manufacturing of designs through the use of CAD models and digital plans in conjunction with numerically controlled machines. Traditionally, digital models help create working drawings, which are then manufactured during secondary processes where the drawings are re-interpreted by the creator of the product or part. With digital fabrication though, the 3D model is used to provide digital information directly to the manufacturing tool, such as a laser cutter, CNC mill, or 3D printer (Dunn, 2012). Since the model is created through purely digital means, every aspect of the model is recorded digitally and is constructed according to those records, so there is no possibility for human misinterpretation. The use of digital fabrication techniques also allows for high customization for manufactured designs. Where traditionally, processes, molds, and other systems of manufacture must be developed specifically for every part, digitally manufactured designs can update and adapt to new digital plans, instantly. For instance, in order to fix a design flaw in a part that has been 3D printed, only the digital model must be altered. If the same part was injection molded and required a design flaw to be fixed, the mold must be physically altered, an expensive and time-consuming process. This increased ability to manufacture customized goods has allowed for the rise of mass customization, where end users can dictate the design of their personal product.

These systems can be seen already in use by companies such as Nike with their Nike ID system and computational design company, Nervous System with their customized jewelry and puzzle products.

Digital manufacturing tools are also becoming increasingly cheap, providing small scale manufacturing opportunities for DIY users and community production centers. While digital manufacturing processes are relatively inexpensive, they typically cannot provide the mass production capabilities of traditional manufacturing processes. Some digital manufacturing tools, such as 3D printing, can also create parts and models that are not possible through any other manufacturing process. Since 3D printing is an additive process, typical design concerns such access for tooling, creating draft angles, and eliminating undercuts are no longer necessary. There are still special concerns for producing successful 3D prints, but generally, the geometric limits of what can be produced are still nearly removed.

There are, of course, more pros and cons to digital manufacturing and each specific tool has its benefits and limitations, but the overall trend allows for a quicker connection between designer and manufactured product, which may be again improved by the use of algorithmic design techniques.

Chapter 3: Guidelines for Algorithmic Modeling

Overview

This chapter will provide a set of guidelines which will help industrial designers implement algorithmic modeling into their design process. These guidelines have been formed based on the preceding research into the tools and techniques of algorithmic modeling as well as case studies involving its use and hands on experience developing algorithmic models. It is important to remember that these are merely guidelines though, and that every design situation is different and presents its own challenges. Algorithmic modeling is a niche technique which cannot, and should not, be used in every situation. This being said, the guidelines are designed to help answer when and how algorithmic modeling techniques should be applied within the context of the industrial design field. Within these guidelines, the reader will also find a brief overview of common algorithmic techniques and terms that can help designers implement their concepts.

The guidelines presented in the following section can be thought of as a timeline of activities and decisions that the industrial designer must undergo in order to successfully use algorithmic design as part of their design process. Generally, the timeline follows the path shown in Figure 15. The process is, first, the evaluation of the problem or design prompt at hand and the development of a geometric or form solution, then an evaluation of the design and situation using specific considerations and a decision to use algorithmic modeling, then form abstraction from a concept to a geometric set of instructions, implementation of those instructions into a modeling system, and, finally, integration of the algorithmic modeled concept or part into the full design solution. By following this process, the industrial designer should be able to successfully integrate algorithmic designs into their work in order to save time and effort or be able to evaluate a situation so that they can jump directly to other methods of modeling, avoiding algorithmic processes altogether.

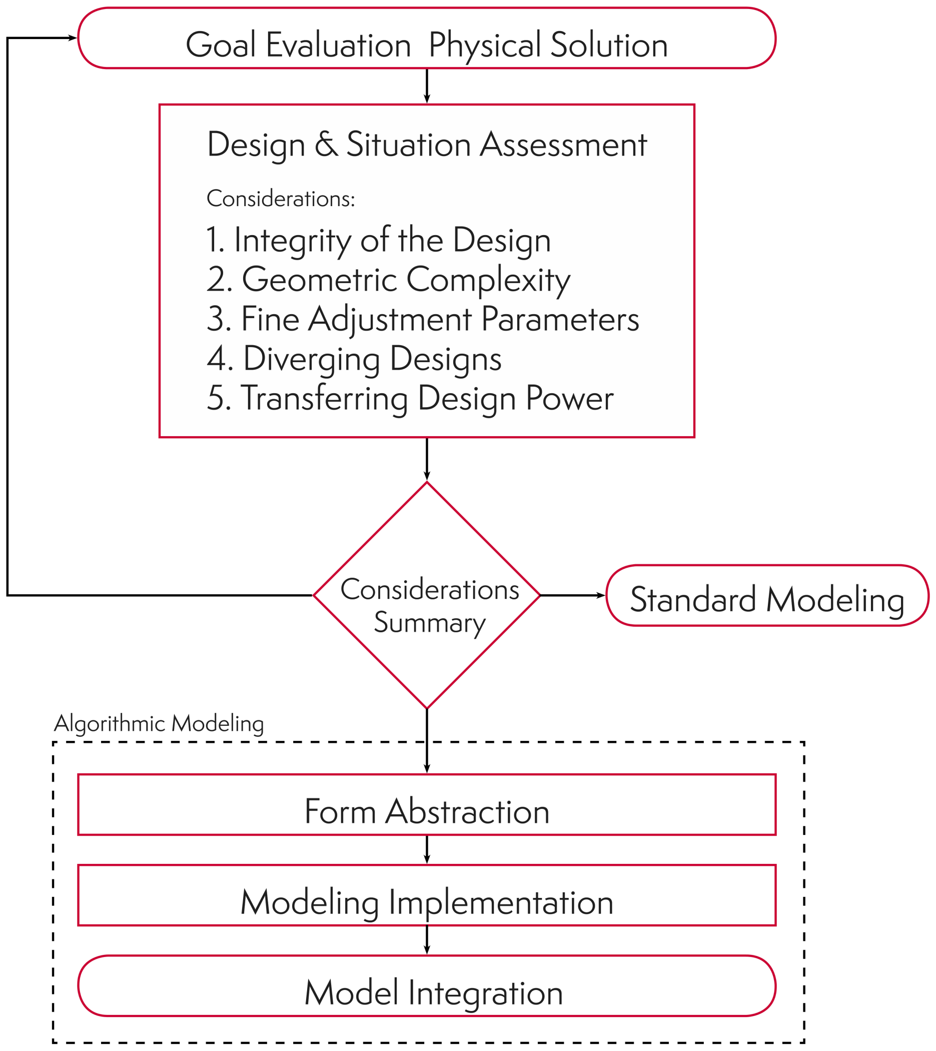


Figure 15: Algorithmic Design Process Guidelines

Goal Evaluation & Physical Solution

The first step of the Algorithmic Design Process Guidelines does not require any algorithmic work; it only requires the general ability of industrial designers to evaluate the requirements, limitations, and goals of their given charge. It is then assumed that from a properly evaluated design problem, a physical solution can begin to develop through the many techniques and methods available to industrial designers, such as sketching, physical prototype modeling and other methods of generating form concepts. Algorithmic design can work in many contexts; however, this document is focused on algorithmic design and modeling that is most closely associated with physical geometries. Therefore, best utilization of algorithmic modeling and these guidelines requires ensuring that the solution for a given design situation requires a physical form.

Design & Situation Assessment

Before the question of "how" a physical design should be created using algorithmic design and modeling should be discussed, it must be decided if the situational benefits of using algorithmic design outweigh the cost of using it. Using algorithmic modelers can be a time intensive process even for experienced users. Those educated in the use of traditional 3D solid parametric modelers such as Solidworks or Fusion360 may find the visual programming languages more difficult to navigate and more difficult in which to define their concepts thoroughly. Also, wherein the "sculpting" nature of traditional modelers can provide a sense of progress for the users as their models are formed through the additive steps of extrusions, fillets, and other manipulations, algorithmic modeling processes often don't provide a finalized representation of the form until the entirety of the work has been completed. This can make the design process murky and mysterious to both the designer and to interested parties, such as clients and team members.

The next five topics of discussion are all key considerations to take into account when determining if the design situation in question would be an appropriate use-case for algorithmic design modeling. This is not to say that a design falling heavily under one area of consideration or under multiple areas of consideration absolutely necessitates the use of algorithmic modeling. Again, every design situation is different, and these considerations are meant to give the designer pause to reflect and evaluate the design itself, the system in which it will operate, and their own modeling skills before jumping into a new design process.

Consideration 1: Integrity of the Design

In the current industrial design process, after the designer has performed the necessary research for the formation of their concept, they will begin developing form model drawings on paper or a digital medium. Sketches provide a loose, fast, low-cost method for testing out ideas. The industrial designer will be able to further and further define their model through their sketches until a final form is suitable enough to move into the next step, CAD. The process of moving into the 3D, computer-realm and away from 2D drawings is an assertion from the designer that this version of their concept is properly formed enough to justify the time requirements of CAD. This does not mean that experimentation is completely out the window yet, of course. There is still plenty of opportunity to change the design by adjusting lengths, adding chamfers and fillets, and adding any number of features that can alter the model. The general strategy behind CAD modeling, though, dictates that the designer's modeling process moves from the general to the specific, with large forms being defined first and the details being added afterwards. This is all to say that the industrial designer is typically confident that the general design of the model will be a shaped certain way as the CAD process begins; it's much easier to change the small details of a CAD model than it is to change the larger form shapes once the design is complete. For example, it would be nearly impossible to model a whole product, which has an overall circular shape, and then attempt to change it to an overall square shape while still retaining the detail work of the model. To do this, the designer would likely need to restart the modeling process entirely.

The common-enough sentiment in the design field, "Don't jump into CAD too early," is a warning against designers spending too much time in CAD when their design development has not been not fully formed. This sentiment is, perhaps, even more relevant when speaking of algorithmic modeling. In algorithmic modeling, every aspect of the model is defined, in detail, leaving little room for the designer to attempt to experiment with the overall concept or form they have in mind. Once again, this does not mean that there is not opportunity for the user to make changes to their design. In fact, the heightened parametric power of the algorithmic modeler makes it a great tool for making adjustments to the model and discovering novel designs, which will be discussed in Consideration 3. However, algorithmic modeling is heavily based upon building relationships between items within a system, so unless they are built with care and forethought, algorithmically defined networks such as these can make it difficult to perform significant changes upon the overall design at a later time. The defined relationships are often situational and tied to a specific function or geometric entity. If that entity or tool is fundamentally changed, such as our example of moving from a circle to a square, the-relationships built upon the logic and foundation of a circle may now be ill defined, rendering the model broken. Gaining an understanding of what aspects of a model can and cannot be changed easily later in the design process requires experience using algorithmic modelers and a deep understanding of the tools and functions at work. Even with this understanding, drastic changes to the design intent can be difficult to work with, so it may be better to err on the side of caution when using algorithmic modelers. A note here: changing the overall design intent of a model after working on it is different than making branching large adjustments, which will be discussed in Consideration 4.

If it is assumed that the designers' intent is to create objects and complete models in a timely manner and is not working in an experimental fashion, it can be recommended that before the designer decides to use algorithmic modeling, they should consider whether or not their concept is defined enough to properly abstracted into mathematical terms which can then be implemented into algorithmic modeling systems. If the general direction of their design is not fully fleshed out, they may consider going back to the drawing board to continue their design process before proceeding. The more clearly they can describe the geometry of their design, the more easily they will be able to dictate that design to the modeler. It may be wise here to remind the reader of the dichotomy of algorithmic forms described in *Expressive Form* (Terzidis, 2003): the invention and the discovery. The "invention" side still follows the advice of Consideration 1 because it requires the designer to be able to fully express their intent of their design to the system. For the "discovery" side, better known as generative design, the details of the design are not fully defined, so it may be tempting to think that Consideration 1 is less of a requirement. However, generative designs still require that designers have a full understanding of the limits, forces, and goals which they are trying to work within.

The designer should also be aware of which portions of their design will undergo algorithmic processes. Algorithmic modelers can define a geometry in full, but it may save time to model the product through standard CAD processes and only model select portions of their design, such as a particular part or surface, through algorithmic modeling.

In summary, if designers believe that their design or concept is near-fully formed and that they can confirm the overall design and design details with some certainty, they might consider using algorithmic modeling. This does not yet mean that their design should be used in algorithmic systems, just that having only a general idea of their design may preclude them from using algorithmic design entirely. Again, if the design is only a partially formed idea, still in need of development, the designer should wait to begin using algorithmic design.

Consideration 2: Geometric Complexity

Geometric complexity is a relative phrase that may be difficult to define. Traditional 3D solid-modelers are, indeed, capable of producing complex models of products or buildings with many parts, interactions, and forms, which one could say are complex, in comparison to primitive forms, such as cubes or spheres. When referring to the geometric complexity which algorithmic design is known for, it is the complexity in the definition of the individual forms and not the complexity of the overall design. Algorithms allow the designer to easily define details and change parameters in ways that would take massive amounts of time and effort in standard modelers.

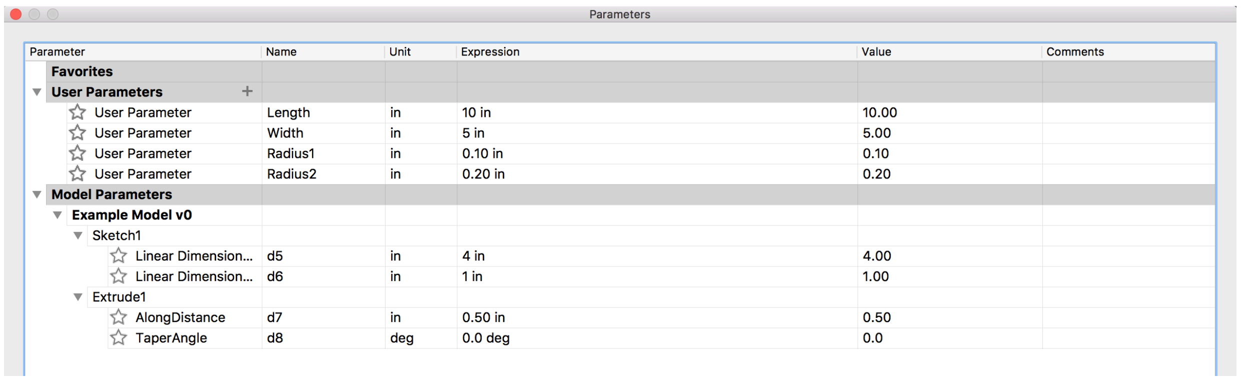
Tiling patterns, interweaving surfaces, fading structures, data-driven-designs, and random designs are all examples of geometric complexities that can be created in algorithmic modeling. Standard modelers do have the capacity to create rudimentary patterns, usually circular, rectilinear, or even patterns along paths, but these tools are often limited in what features can be patterned and can be resource-intensive actions for the modeler to compute.

Geometric complexities can serve functional or aesthetic purposes for the designer depending on their needs. Complex geometries can be applied to forms to give them new physical characteristics: making them lighter, more porous, or more textured. These physical changes may allow the form to operate better as a product or satisfy other design criteria. This document will not discuss the theories and complexities of design aesthetics other than to say that algorithmic designs and the complexities that can arise from it could be regarded as visually stimulating or, at the very least, novel. This aesthetic feature may be leveraged to make a product more appealing to an end consumer even if it does not serve any functional purpose.

In conclusion, if an industrial designer would like to implement complex geometric forms into their design for whatever purpose, it may be beneficial for them to consider using algorithmic design to define these forms since creating the same forms in standard 3D solid modelers may be incredibly time intensive, if not impossible. If the designer believe they can create their design in a timely manner in a standard modeler, it may wise to do so.

Consideration 3: Fine Adjustment Parameters

In a standard 3D modeling environment, the designer will usually create sketches, define the details of these sketches using dimensions, and then use those sketches to create solid forms through extrusions or other similar methods. Values like these dimensions, extrusion lengths, radii, and others are all parameters that are saved within the system. In a parametric modeler, the designer can go back and edit the values of these parameters, and the system will attempt to update the solid model with this new information. Changes within these values will be referred to as "fine adjustments" in that changes to them do not change the internal logic of the system or the structure of the flow of information through it. Parameters can be changed individually when needed, but sometimes defining single parameters that are used in multiple places can provide a much faster, easier to manipulate system when changes are required, such as the parameter system found in Fusion360. This system lets the user create general parameters and define a value for them. For example, the designer could set "length" to be equal to 10mm and can simply enter "length" when prompted with a value entry box for a dimension, or an extrusion distance. If the "length" parameter is then changed to be 20mm, the model will update all of the parts accordingly.



Parameters

Parameter	Name	Unit	Expression	Value	Comments
Favorites					
User Parameters +					
☆ User Parameter	Length	in	10 in	10.00	
☆ User Parameter	Width	in	5 in	5.00	
☆ User Parameter	Radius1	in	0.10 in	0.10	
☆ User Parameter	Radius2	in	0.20 in	0.20	
Model Parameters					
Example Model v0					
Sketch1					
☆ Linear Dimension...	d5	in	4 in	4.00	
☆ Linear Dimension...	d6	in	1 in	1.00	
Extrude1					
☆ AlongDistance	d7	in	0.50 in	0.50	
☆ TaperAngle	d8	deg	0.0 deg	0.0	

Figure 16: Fusion Parameter System

The ability to control these parameters lets the designer continually update and modify their design. Often though, in practice, it can be difficult to see how changing a parameter will affect the model if the system has grown too complicated. These relationships can even break the model if changes that disrupt the flow of logic are made.

Since algorithmic modelers use linked relationships to define the geometry of the model, it can be easier to see exactly how a parameter will affect the design than in standard modelers. It could even be argued that the act of having to physically link values to functions may provide a more in-depth understanding for the internal logic of the model. The way in which algorithmic modeling VPLs show parameters as adjustable entities and their connections to the other entities, which they drive, are invaluable to the designer's ability to more easily update and change their system. VPLs such as Grasshopper and Dynamo also have added functionality such as number sliders, data set inputs, and organizational tools, which can give the user a greater ability to quickly experiment and prototype with their model. These fine adjustments can create an infinitely variable geometry.

In summary, the amount of fine tuning to be made to the designer's model or the number iterations of designs needed should be a consideration for the designer before they begin using algorithmic modeling. For example, if the design situation has very specific, absolute dimensions without need for form experimentation, it may not be wise for the designer to proceed in using algorithmic modeling. If the designer is unsure of the specific dimensions, or is open to experimentation in the form, they may be well served to use algorithmic modeling over traditional modelers. This is not to say that traditional modelers such as Fusion360 or Solidworks cannot provide fine parametric adjustments such as these, just that the nature of VPL algorithmic modelers can provide faster, easier adjustments.

Consideration 4: Diverging Designs

Generally, in an algorithmic modeling VPL such as Grasshopper and Dynamo, the logic of modeling runs from left to right, with strings from outputs on the left informing inputs of functions on the right, which in turn create their own outputs. A similar logic can be found in standard 3D modelers used by industrial designers, which have a kind of function history, where entities are formed and organized based on when they were created and how they relate to other entities.

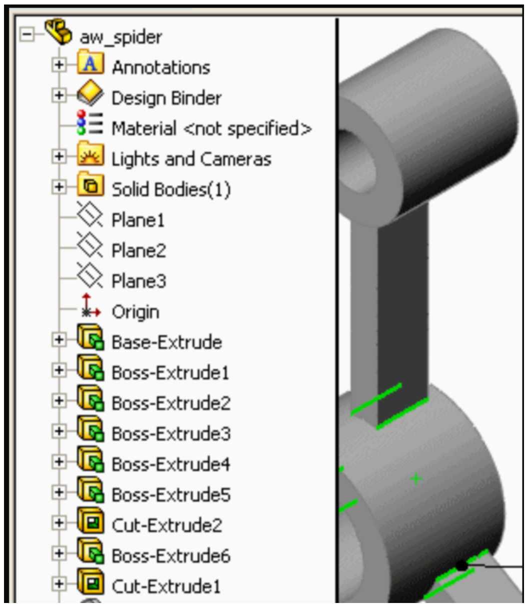


Figure 17: Solidworks History Tree

While these systems are sometimes referred to as trees, they often operate more like timelines, with a linear flow of logic where each item is built upon the last. Changes early in the timelines can have great effects on the rest of the model, making it difficult to make significant changes without disrupting the rest of the work done. If large changes are to be made, another separate version of the model is usually saved and becomes an independent entity, unlinked from the original version.

In an algorithmic modeler, there is the logical flow from left to right, but there is also room for expansion, vertically. The outputs of a certain function can be used to inform wholly different areas of modeling which can operate independently from one another. If the design-needs dictate that one set of functions be used in some cases but not others, entire processes can be added or removed from the logic flow without breaking the system.

These branching modeling systems allow for many designs to be available in one system without the need to create different, separate, saved versions of a model. Where fine adjustments allow the designer to create small parametric changes to their design by adjusting values and parameters, a diverging design can create large changes by inserting or removing branches of function from the modeling logic. These changes can drastically alter the design of the model and can provide great opportunities for experimentation without removing or altering the original modeling logic or opening a different modeling system.

If the designer believes that their model may require additional updates in the future, which can be implemented as an alternate branch to their original design, they may consider using algorithmic modeling. In the field of industrial design, the ability to create alternate versions and iterations of models is beneficial, and algorithmic VPL modeling can make this process easier to visualize and execute. If the design is a singular product which won't be updated in the future with different options, standard modeling may be the preferred choice.

Consideration 5: Transferring Design Power

The rise of mass customization, giving users more control over the design process, requires a method of rapidly changing designs to suit users' needs or to give the customization power to users directly through some sort of interface. Both of these situations can be fulfilled more easily with algorithmic modeling than compared to traditional modeling software. The customization could be as simple as a small variable change along a specific parameter, or as complex as implementing a wholly different paneling mechanic on a surface through a diverging design. There are also situations where data-driven designs can be created which import whole data sets to be used as inputs for specific functions. For the designer, the requirement for customizations, and therefore rapid iterations, is fulfilled by parametric power of algorithmic systems, previously discussed in Considerations 3 and 4. However, the ability to provide the user with an interface in which they can experiment with parameters to create their own custom designs can be done more easily through algorithmic systems. In this scenario, the designer is responsible for not only creating the functioning model system, but also determining which parameters and controls the users will be able to control in an interface.

Additionally, if the designer could potentially hand off a design to another designer or coworker, using a VPL algorithmic modeler is a more transparent design environment than a traditional modeler. This may allow alterations to be more easily made by this secondary party.

In conclusion, if the designer intends on creating a system wherein individual users or those not involved in the original design process will have the ability to customize and dictate designs, it may be recommended that they use algorithmic modeling instead of traditional modelers.

Considerations Summary

By reviewing these considerations of the particular design-situation at hand, the designer should now be able to make an informed decision into whether or not they should proceed into the use of algorithmic modeling and begin form abstraction.

Again, all or just one consideration could apply to the situation, but this doesn't affect the decision to use algorithmic modeling, necessarily. For instance, a design could be fully formed, with only one iteration needed, with very defined parameters, with no alternate designs, and no need to transfer the design power, leading the designer to believe that they do not need algorithmic modeling, but if the design is also suitably complicated, this one factor may push the designer towards using algorithmic design. If they have decided against using algorithmic techniques, they should begin using standard modelers to do their CAD work.

Form Abstraction

If the industrial designer has decided that the physical design and design situation necessitate the use of algorithmic modeling, the designer must now begin the process of transferring concept ideas into a form which algorithmic modeling VPLs can read and understand. The translation between a concept on paper and a series of logical steps that define the form is a difficult one, but with enough practice and understanding, can be manageable. Abstraction is "the process of removing physical, spatial, or temporal details" from a system. (Colburn & Shute, 2007). This allows the designer to think through the discrete steps of the algorithm without being bogged down with the many details they will have to define in the VPL. This also allows the designer to focus more on the interactions between the elements, ensuring that the designs can work in any number of situations, not just a particular one.

For instance, a designer may have decided that the best course of action is to model a cube, but sometimes an algorithmic modeler can define identical geometries in different ways. A cube could be described as a cuboid where length, width, and height are all equal to the same given value, all centered with a corner at point (0,0,0). A cube could also be described as a series of 6 ruled surfaces with shared points between lines along sides of each surface. The abstraction of geometry requires an understanding of the vocabulary and mathematical descriptions that algorithmic modelers use to define forms. If the concept cannot be at least partially abstracted by the designer and put into terms that can be found in algorithmic modeling VPLs, the designer may find it difficult to begin the implementation process. This idea is not too foreign to the designer as they already use abstraction to form their traditional cad models, changing thinking from the physical designs to terms like “extrusion”, “sweep”, “loft” etc.

Merely thinking through which tools will be used in the VPL is an abstraction, but it could be helpful to sketch the different steps on paper, without diving too far into the details of each step. The upcoming section, Algorithmic Design Methods and Tools, will include definitions and concepts that can help the industrial designer understand how to better define their concept for the implementation process, and the Abstraction portion of Chapter 5 will provide an example of an abstraction.

Modeling Implementation

Now that the geometric goal has been defined, and that geometric form has been at least somewhat abstracted into mathematical terms, it is now time for the designer to begin using a visual programming language for the purposes of algorithmic modeling.

There are two VPLs that this document focuses on, Grasshopper and Autodesk Dynamo. There are differences between the two, but for the general purposes of modeling using algorithmic design, they operate in a very similar fashion. The exact ways that a designer should go about modeling the concept in Grasshopper or Dynamo cannot be fully described, since every situation requires specific nodes and connections to be made. It is similar to describing to someone “How to cook” or “How to use Solidworks.” The best solution to these general problems is to become familiarize with the tools and techniques at hand and view common use cases of how they are implemented.
Once properly familiarized with the tools or have found a use case that resembles the design direction to be followed, the designer can then begin to construct the algorithm that follows the abstraction. In Chapter 4, Algorithmic Design Methods and Tools, specific terms, tools, methods, and techniques, which the reader can use to help with the construction of their algorithmic designs, are described.

Model Integration

Once the design abstraction has been fully implemented into a VPL and computed, the VPL will show a rendered version of the design. This is not the end of the design process, however. The VPL algorithmic design must be extracted and placed into a standard modeler such as Solidworks or Fusion360 or Rhino so that the designer can continue working around the model to make it into a full product or prepare it for rendering or digital fabrication. This process is different depending on which VPL being used and which programs to transfer the model to. For instance, in Grasshopper, one can just “Bake” their design directly into the Rhino workspace they are using. Alternatively, the designer could use exporting nodes which take in geometry as inputs and export to a specified file location. For example, the Dynamo Export to SAT node takes in geometry and saves it as a .sat file, which can be opened in Solidworks or Fusion360 along with many other standard modeling software.

The integration step can sometimes cause problems in the algorithmic design process that the designer should be prepared for. Sometimes surfaces won't bake properly, sometimes the model will be unintentionally scaled, or the model could have surfaces missing. The designer must troubleshoot these problems and find work-arounds to get to the final form or file that they need.

Chapter 4: Algorithmic Design Methods and Tools

Due to the sheer number of tools and features available for use within algorithmic modelers, not all of them can be described within this text. What can be described are general geometric terms, which have associated components that are easily searchable within Grasshopper or Dynamo for use within designs. For demonstration purposes, often a Dynamo node will be shown, instead of a Grasshopper node, because the Dynamo nodes are more clearly labeled with a title, inputs on the left side, and outputs on the right side. Additionally, some common algorithmic techniques which may work as a starting point for new designs are also described.

Geometric Terms

Points

A point is zero-dimensional primitive meant to describe a unique location within a space. Points have no volume, length or area. By themselves, points do not describe any solid geometries, but provide references for additional items such as lines, planes, surfaces etc. They are the most basic of building blocks for defining geometry and are used widely through algorithmic modeling.

Points are most commonly defined through the use of coordinates, a set of numbers that define a location in a coordinate system. In two dimensions, points are commonly defined by coordinates in a Cartesian (x,y) notation, where the point lies an x distance in the x-direction and a y distance in the y-direction. A similar system exists for a three-dimensional coordinate, where an added z-direction is introduced resulting in an (x,y,z) notation. Note the general use of "direction". Generally, coordinate systems will follow axes (x-axis, y-axis, etc.) but this is not always the case such as when coordinate systems describe parametric points along surfaces, which may have non-linear u and v axes.

As expected, within algorithmic modelers, points are usually defined through coordinate systems of some sort. Dynamo and Grasshopper both have options to define points as Cartesian, cylindrical, and polar coordinates. The example node shown in Figure 17 takes in Cartesian x, y, z values as inputs and outputs a point. This node can also input lists of x, y, z values to output multiple points.

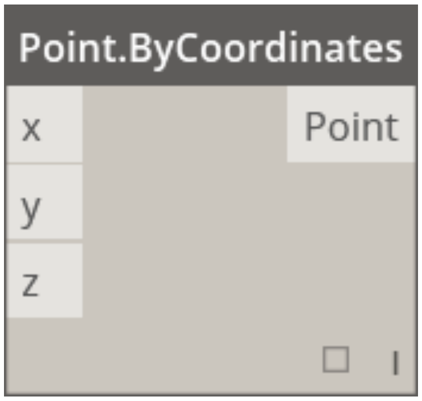


Figure 18: Dynamo node for creating a point with Cartesian coordinate inputs.

The Point.ByCoordinates node assumes that the coordinate system has an origin at (0,0,0). If a different coordinate system is required, there exist other nodes that operate in the same way, but require an input coordinate system.

Vectors

A vector, more specifically a Euclidean vector, is a mathematical representation of an object that has a magnitude and a direction. In algorithmic modeling, vectors are often used to help define a direction for other geometries, such as defining an extrusion direction. By themselves though, vectors are not solid geometry. Vectors can be created in multiple ways, but can usually be extracted from straight-line geometries. The node shown in Figure 18 creates a vector from the origin point (0,0,0) to a point at a defined (x,y,z).

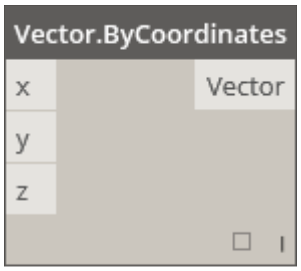


Figure 19: Dynamo node for a creating a vector using a coordinate point.

Curves

For both Autodesk Dynamo and Grasshopper, Curve is used as a generic term to describe curves and lines of all sorts. Curves can be straight lines, arcs, circles, splines, polygons, NURBS Curves, polycurves and more. Anything that one might define as a sketch entity in another CAD program is a Curve. A Curve is more accurately defined as the infinite number of points that can be found by plugging t into a set of parametric equations, such as $(x = 3t,\ y = 5 - t)$, where $0 \leq t \leq 1$. This simply means that all

Curves have a starting point and an ending point, even Curves such as circles. Also, these starting and ending points are defined at $t = 0$ and $t = 1$, with every possible point between being defined by $0 < t < 1$.

A straight line is commonly defined as line connecting two points, and while this is the most common way of defining a line and is easily done in algorithmic modelers, it is not the only way.

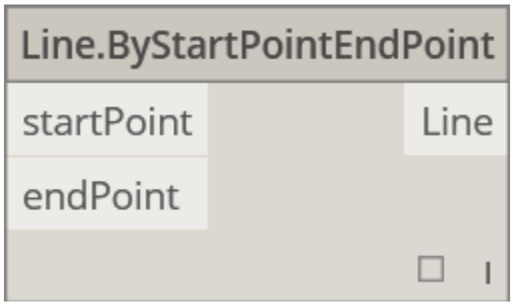


Figure 20: Dynamo Node for a line using start point and endpoint inputs.

Straight lines can also be defined by using a starting point, line direction, and length, or in other non-obvious ways, such as a best fit line through a set of points, or as a tangent line from another Curve at a given point.

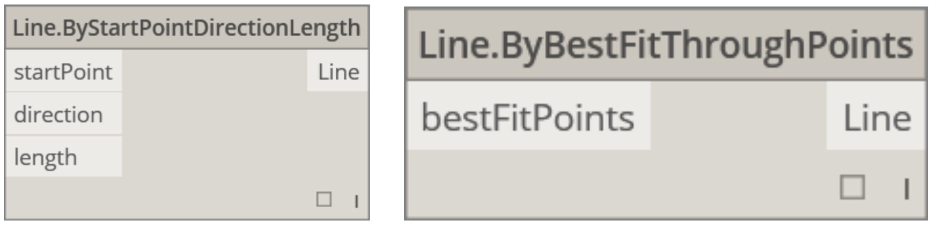


Figure 21: Dynamo nodes showing alternative straight-line creation techniques.

A polyline is a Curve that is made of two or more lines that have been joined together. A polyline that is described as "closed" is one where the last and first points of the polyline occupy the same space. Some nodes in Dynamo and Grasshopper can connect a series of points together to form straight lines between consecutive points, and then join them together to create a single polyline. The decision to make it a closed polyline can be toggled by a simple true/false input.

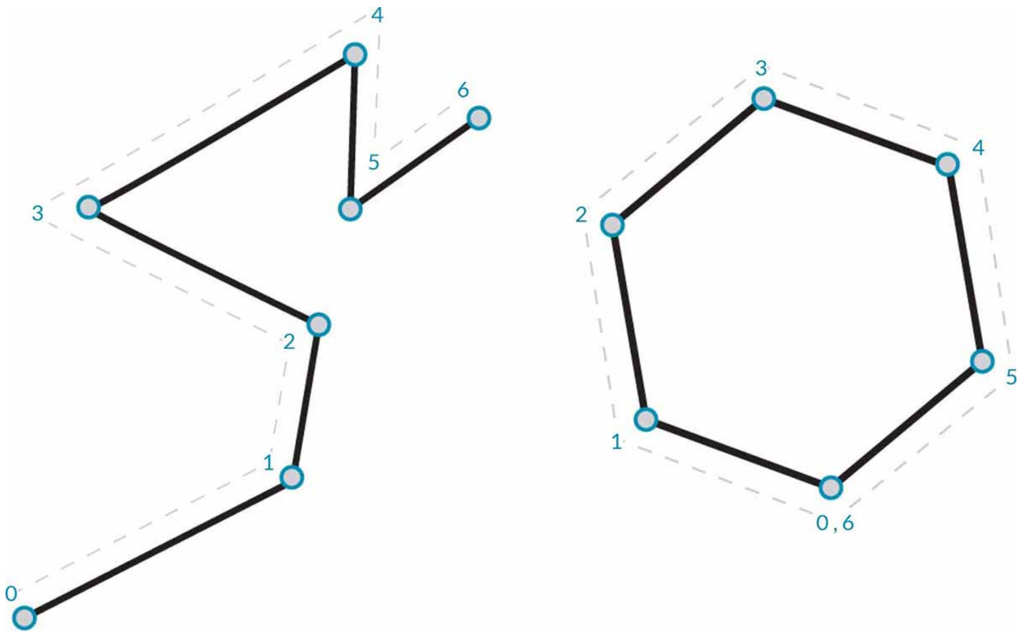


Figure 22: Polylines showing open (left) and closed (right) forms.

Two Curves, the arc and the circle, are very similar in that they usually require a center point and a radius which all of their described points lie on. The arc is only a portion of a full circle, however. There are also nodes and components available to describe ellipses and other irregular arcs without a constant radius.

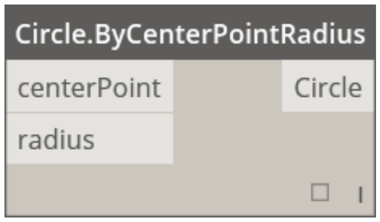


Figure 23: Dynamo node for a circle, with a center point and radius input.

The acronym NURBS stands for Non-Uniform Rational B-Splines. NURBS are representations of curves, surfaces, and solids that are easily transferable to other systems since they are so mathematically well-defined. A NURBS Curve is one that utilizes control points to define how a Curve moves through space. This process is similar to how splines and Bezier curves are defined in standard modelers. The location of the control points, the curve degree, weight values and knots are all aspects of the NURBS curve that define it, and can be controlled in algorithmic modelers.

Planes

A plane is a two-dimensional object which extends in all directions within the dimensions. It can be thought of as an object that describes a geometric flat surface. Planes have an origin, an x-direction, and a y-direction. They also have a z-direction that is normal to the plane surface, but objects that do not share the same z-coordinate as the plane are “out-of-plane” and will not intersect it. Planes do not have any volume and are mainly used to help describe other geometries that can be created in algorithmic modelers.

Surfaces

In the same way that Curves can be thought of as being a continuous string of points between two values, a continuous set of Curves together define a surface. Where Curves are defined by a parametric function using variable t, Surfaces use parametric coordinates (u, v) to define points in space. Again, u and v are both greater than or equal to zero and less than or equal to one. Shown in Figure 24 is how any point on the surface, regardless of whether or not the surface is planar, can be defined by u and v.

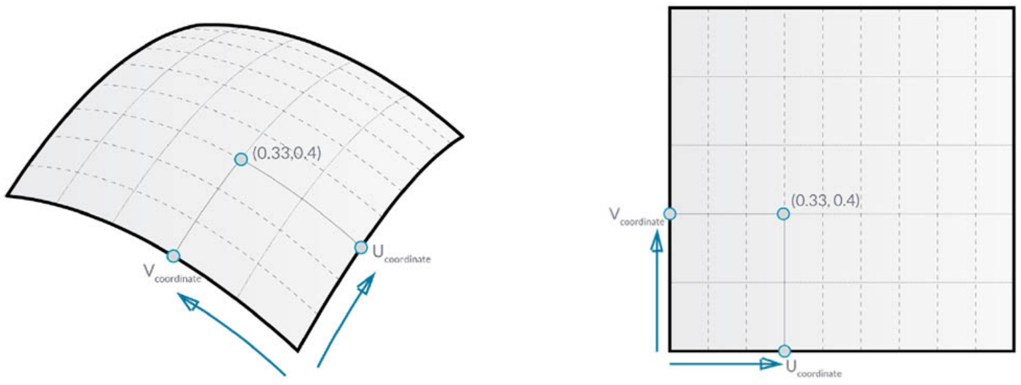


Figure 24: (U,V) Coordinates on a plane graphic from The Dynamo Primer

An isocurve is a Curve that follows along a surface, where either the u or v parameter remains constant while the other parameter encompasses the full domain of values. Isocurves can be thought of as a cross section of a surface at a certain value.

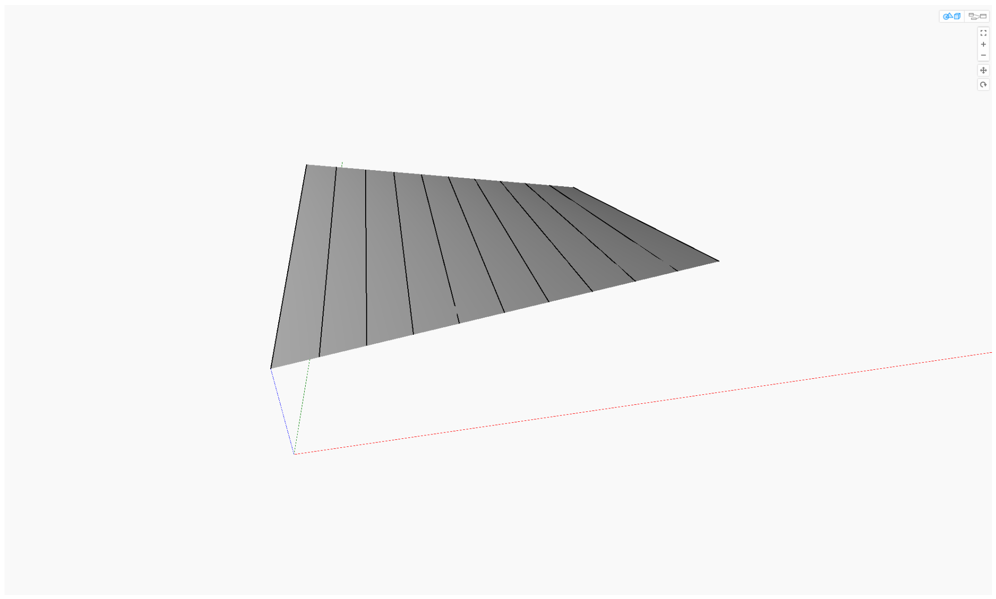


Figure 25: Isocurves on a surface in one direction.

Every point on a surface has additional corresponding data such as normal vectors and perpendicular planes, which can be extracted through some nodes or components. A perpendicular plane is one in which the axes are tangent to the u and v isocurves at any particular point, and the normal vectors are vectors which are perpendicular to the corresponding perpendicular plane.

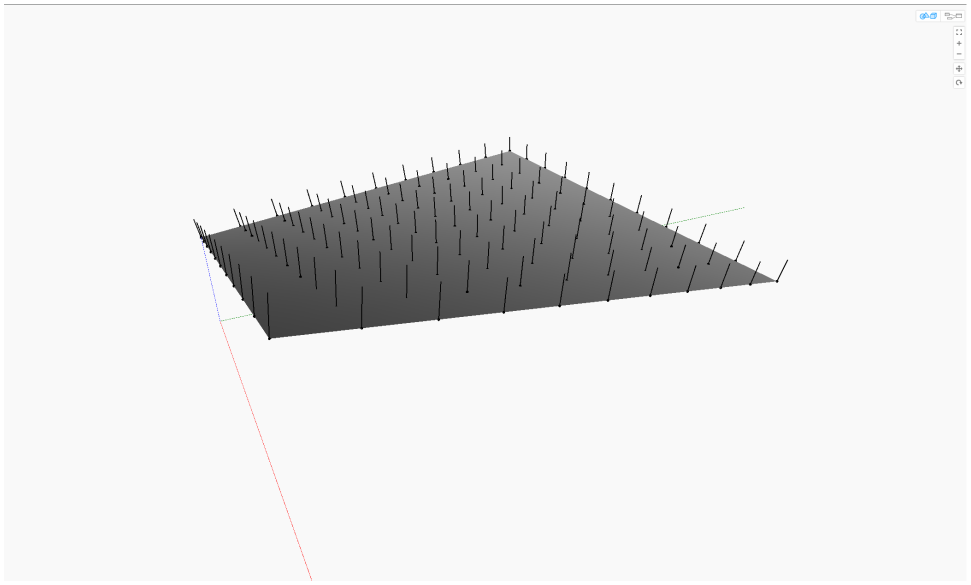


Figure 26: Lines along normal vectors across a surface

As one might expect, a NURBS surface is one that is made up of a network of NURBS curves in the u and v directions. Again the points, degree, and weight of the NURBS Curves can be adjusted to affect the shape of the surface.

Solids

Unlike the geometries previously discussed, solids are geometries that have a volume. Solids are made up of one or more surfaces and have a definitive "inside" and "outside" to them. Cubes, Cylinders, Spheres, and other basics shapes are all solids, along with shapes created from standard modeling methods such as closed extrusions, lofts, sweeps, and revolutions. A solid is made up of different parts that describe its topology: vertices, edges, and faces.

Vertices are the positions on a solid that are occupied by a single point. For instance, a cube has eight vertices, one at each of its corners. A solid does not necessarily need a “sharp” point, though, to have vertices. A cylinder has vertices where the corners of the surfaces that make up the body of the cylinder meet.

Edges are Curves, which are defined by the shape of the geometry. A cube has 12 edges, where the surfaces of each side come in contact with each other. Again, some shapes, which may not appear to have edges, do. At first glance, a cylinder has two circular edges on top and bottom of the cylinder, which outline the “caps”. There is also a third edge where the curved surface that makes up the cylinder body wraps back to make contact with itself again.

Faces are surfaces that make up the sides of solid geometry. This is fairly easy to understand since solids require surfaces to be created in the first place. A cube has six faces and a cylinder has three. There are some instances that are not as clear though, such as a cone, which has two faces, and spheres which are made up of only one face. The different topologies can be extracted and singled out for use in creating other geometries, if needed.

Note that sometimes the terminology surrounding certain features may change depending on the program being used. Dynamo mostly uses the term Solid when referring to solids with the vertex, edge, and face topology we’ve discussed, but Grasshopper may use BREP to refer to the same thing. BREP stands for boundary representation, and is also a solid, which is “bound” by surfaces.

Meshes

Meshes are geometric systems that are made up of many small triangular, and sometimes quadrilateral, surfaces. These surfaces have vertices and faces as part of their topology, which can be indexed and selected individually. Meshes do not require the mathematical definitions of NURBS surfaces so, in general, the control a designer has over a mesh is more limited than that of a NURBS surfaces; however much more complicated geometries can be described by meshes and they are more computationally lightweight. The lack of mathematic definitions means that meshes can be manipulated locally without affecting the whole model. This concept is shown in an excerpt from the Dynamo Primer, shown in Figure 26.

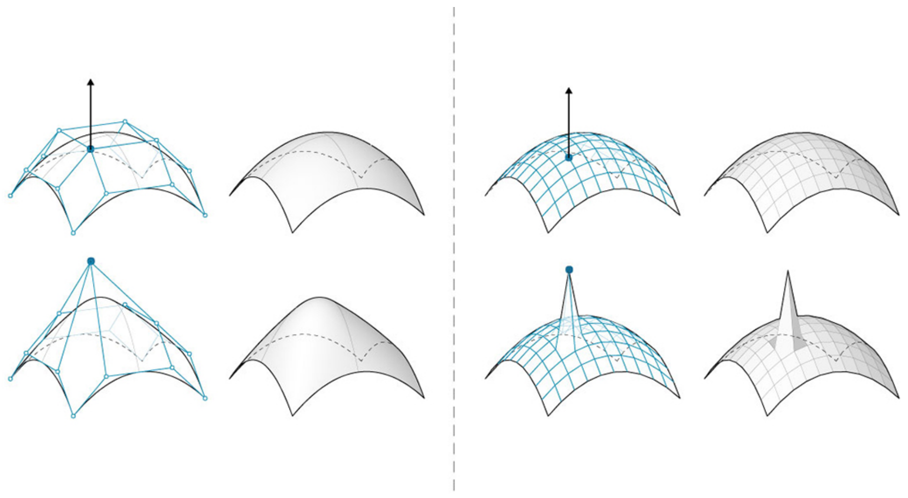


Figure 27: NURBS surface showing how control points affect the whole system (left) and a mesh surface showing how edited control points only affect the local faces (right).

Data Management

Inputs

Algorithmic design and visual programming languages rely on the exchange and manipulation of data between nodes and components. Algorithmic tools usually require specific inputs in order to operate correctly, and there exist many types of inputs which can be used in order to build a model. Nodes take in inputs, and produce outputs, which can then be used as inputs on other nodes, like the set shown in Figure 28.

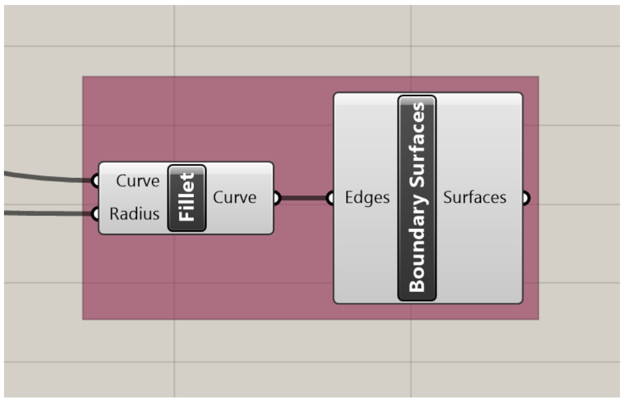


Figure 28: Nodes & Connections in Grasshopper

Numbers are the most widely used inputs in algorithmic modeling. This is understandable since numbers are used to define dimensions, distances, sizes and other factors in standard solid modelers. It is important to note here that sometimes nodes and components may require an integer value to operate. An integer is a whole number such as 1, 12, 100, as opposed to a number with a decimal, such as 1.2, .5 and 10.2. Algorithmic modelers have several options to input numbers, such as static number inputs, number sliders, dials, and more complicated routes like using a connected excel sheet to define values.

A string is a type of data that represents text. Strings are used heavily in many types of coding systems; they're even the basis for the traditional "Hello World" coding lesson, but they may not be as important for 3D modeling. There are still situations such as creating labels or cutting in decals where the designer may need them, though, and algorithmic VPL modelers do have tools to accept strings as inputs.

Color is another input available for use, and images can be used as input in VPL algorithmic systems. There exist nodes and components that can pull in images and sample their contents to determine the color and brightness of a photo at different points. Color information and associated numbers can then be used as data to inform other components to create geometries and perform other functions.

Algorithmic modeling centers around geometric modeling techniques; therefore it should be no surprise that different geometries are actually data types to be transferred from one component to another. The previously listed geometries such as Points, Curves, Surfaces, Solids, Meshes and more are all different types of geometry which can be accepted as inputs for other tools. This also means that geometries that are created in other programs can be pulled into Dynamo or Grasshopper to be used as inputs.

Lists

A list is an organized collection of items. In algorithmic modeling, lists contain data types like those listed above. A list can be full of numbers, strings, or geometries. A list can be just one item, many multiple items, or even contain other lists.

In order to understand how a list operates the designer needs to know where items are on the list. The position of an item is called its indices. The numbering convention of indices may be strange to those unfamiliar with coding languages, since it doesn’t begin with one, but starts its numbering at zero. This means that the first piece of data in a list is always at index 0. Items in lists and the list itself can have many functions performed on them. Items can be extracted at certain indices; lists can be counted, can be reversed, chopped into pieces, shifted, truncated, combined with other lists, filtered to only allow certain values through, and any number of other alterations that can be combined to change the contents of the list to match the requirements of the current system being designed.

When using two lists as inputs into a function, such as multiplying two lists of numbers together, unexpected problems can occur, especially when the lists have different lengths. How can a list with three numbers be multiplied with a list that has four numbers? The answer is by defining the list lacing. Lacing is the way in which items in lists interact with items in another list. There are three types of lacing: Shortest List, Longest List, and Cross Product. Shortest List lacing takes the shorter of the two lists and only provides interaction for that number of items. This can be seen in Figure 29 where excess items in the longer list have no interactions with that of the first list.

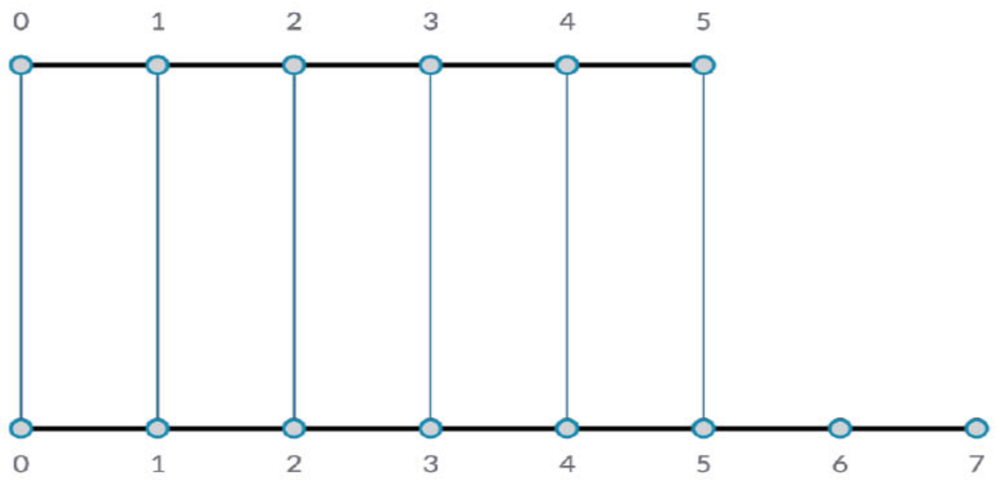


Figure 29: Shortest List Lacing

Longest List lacing determines the number of items in the longest list and provides interaction for all of those items. This means that the last value in the shorter list will be reused to interact with the remaining values of the longer list. This relationship is more easily seen in Figure 30.

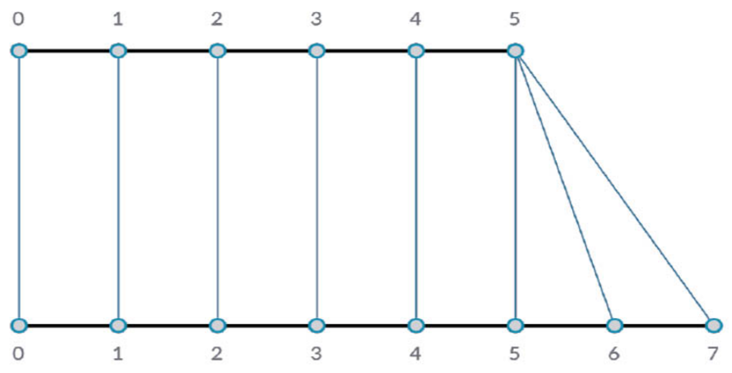


Figure 30: Longest List Lacing

Cross Product lacing is used in cases where the designer would like every item in a list to interact with every item in another list, no matter the lengths of either list.

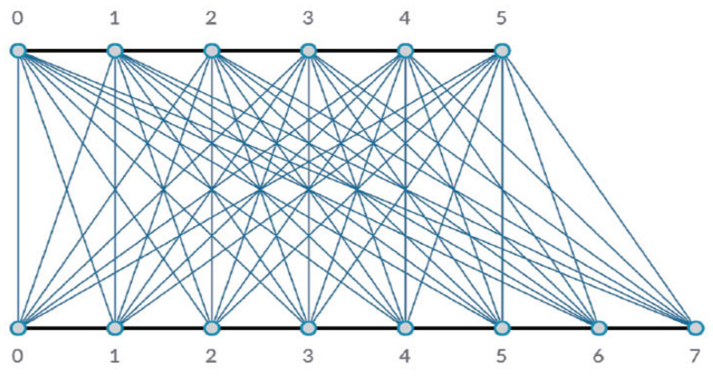


Figure 31: Cross Product Lacing

It is important to understand the structure of lists if one is to manage the data within them. Lists are generally organized into metaphorical trees. A simple list containing ten items can be visualized as a tree (the list) with ten branches (the items). If the list contains ten lists each with ten items in them then the list can be visualized as a tree (the top-level list) with ten branches (the sub lists) with ten twigs attached to each (the individual items). Algorithmic modelers have tools and methods such as level selecting, list flattening, list grafting, which can alter this tree structure in order to fit the needs of the situation at hand.

Algorithmic Techniques

Algorithmic modelers can give unprecedented freedom in creation to the industrial designer. The designer is only limited by the ability to properly dictate a concept to the modeler. However, there are noticeable trends for common algorithmic techniques that can be found in various textbooks, user guides, and real-world implementations. This section will provide a brief explanation and examples of use of these common techniques, largely leaving out the details of the implementation, but perhaps these technique descriptions will show the designer what is possible, or provide a starting point for more in-depth work. These techniques are not in any particular order and can be used in conjunction with one another. Additionally, the examples shown will not include the full coding procedures, but rather general strategies for their creation. The full code for each example is available at: https://drive.google.com/drive/folders/1_iYBf6XisuOJ3Yhe0WXgLgz12W5Bn9bP?usp=sharing

Repetitions

Here, a repetition will refer to copying geometry in a repeating fashion, multiple times. This can be done in traditional modelers through patterning tools, such as rectangular or circular patterning tools, but in an algorithmic modeling system the user has greater control over what can be repeated and in what way. There is also opportunity to alter the repeated geometries so that its properties change on each repetition. The size, orientation, and shape, among other attributes, can be altered to create more interesting repetitions. Creating patterns of geometry in algorithmic modelers often relies on the designer to create a list of positions or changes that then inform a simple tool such as a "Move" or "Scale" component or node.

For this example, a list of coordinate positions has been created to serve as the center for a series of circles. The positions all have different z values, so the associated circles line up in a vertical stack. The radii of the circles are also defined to be associated with list of increasing values. For both the positions and the radii of the circles, a list of changing information is needed to define the circles' characteristics. The information about the circle is created first, then applied to a circle-creating node, where in a traditional modeler, a circle would be created and then patterned in a certain direction.

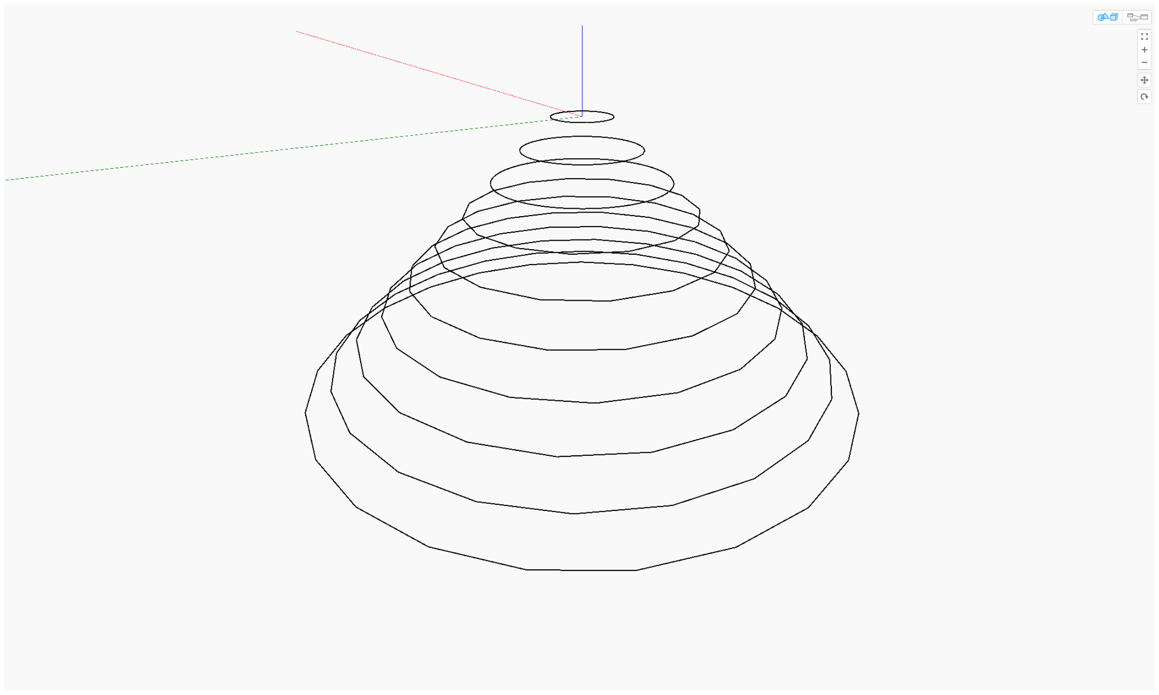


Figure 33: Circles of increasing radii and decreasing z-positions

With the circles now created, they can now be used as the basis for other functions. Next, a polygon function will be used that has two inputs: a circle as the base, and the number of sides the polygon should have. By applying the list of circles created to the polygon function and inputting an increasing list of polygon side counts, a stack of polygons with an ever-increasing number of sides has been created.

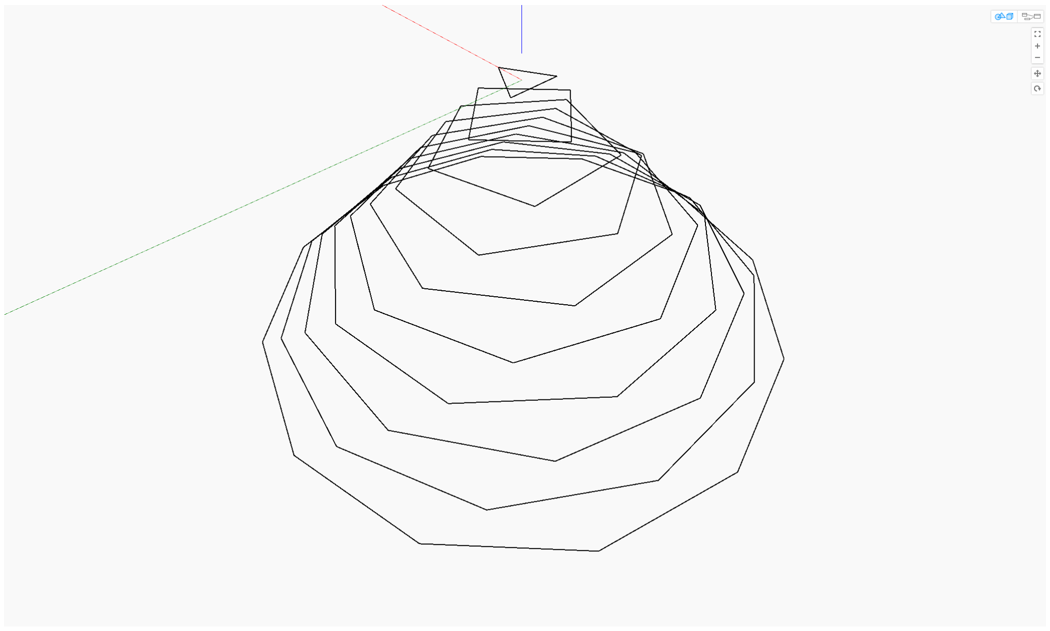


Figure 34: Stack of polygons with creasing radii and side counts with decreasing z-values

Next, the polygon curves will be used to extrude the shape into the negative z direction to form a solid in order to better visualize what has been created.

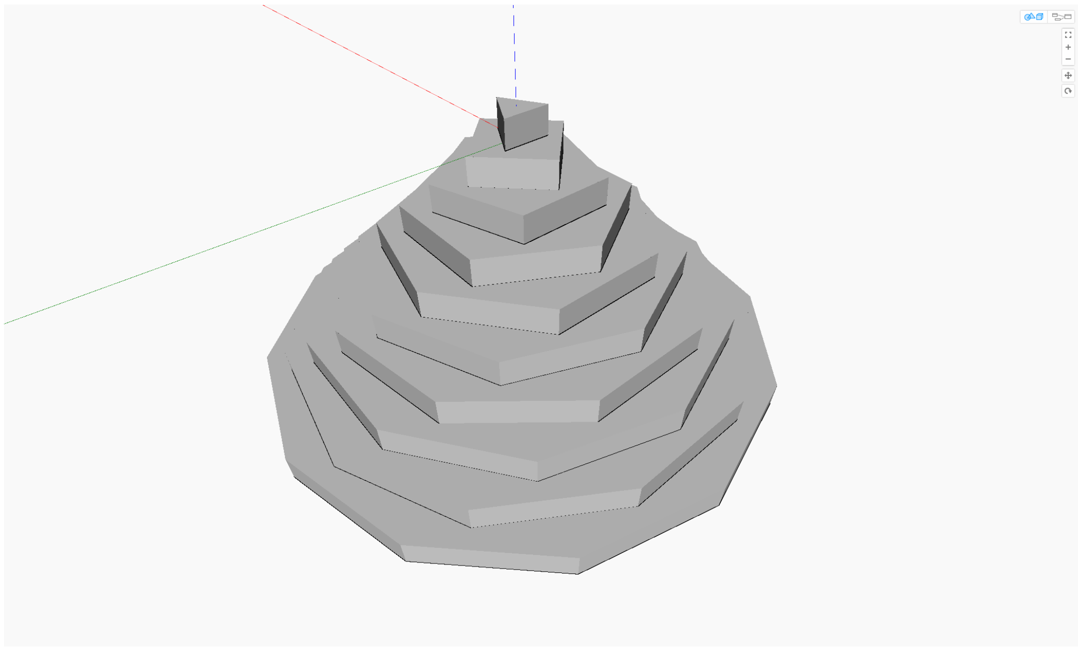


Figure 35: Stack of 10 solid polygon parts

Since the nature of repetitions in algorithmic modeling is that of using lists to define inputs to nodes, if it is necessary to change any aspects of the model only the information in the lists created needs to be changed. In this case the number of initial circles, radii, and polygon sides are all tied to the same input list, so by increasing the length of this list, more and more polygon stacks can be created. Figure 35 shows a stack with 10 units, while Figure 36 shows a stack with 25 units.

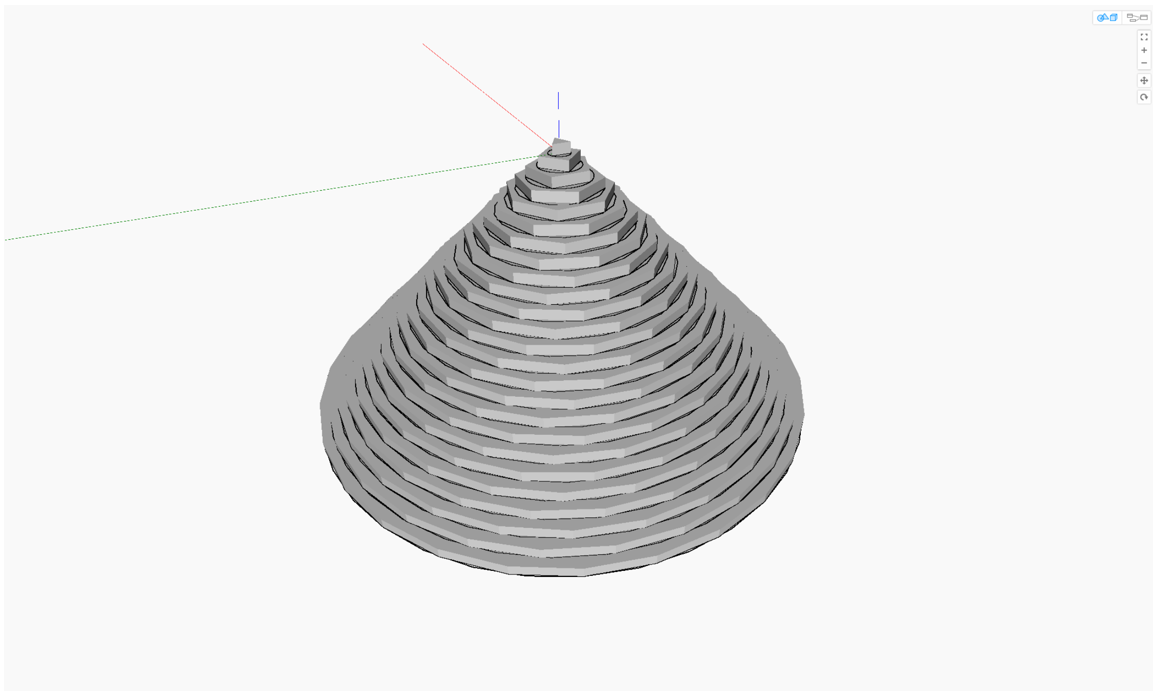


Figure 36: Stack of 25 solid polygon parts

One of the benefits of using algorithmic design is being able to quickly see how designs change at different scales. These polygon stacks may be relatively simple but when increasing the list length up to around 65, seen in Figure 37, the shape takes on a different quality as the polygons converge towards full circles. Again, the only difference between these different sized examples was a single input number. Repetitions such as these can help create interesting visual patterns, provide unique textures to be applied to handles for example, or define form through small changes.

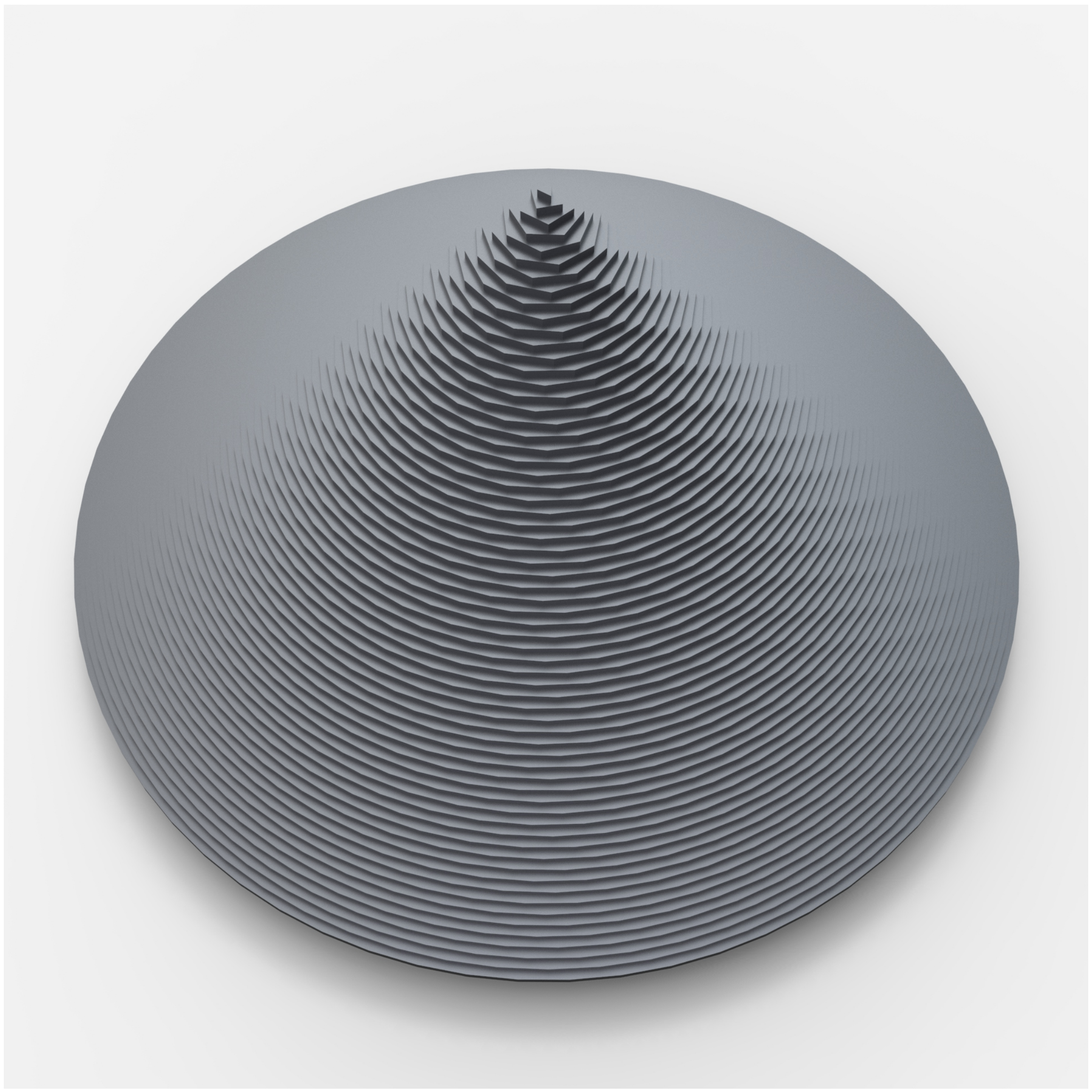


Figure 37: Rendered stack of 65 solid polygon parts

Divisions

Instead of creating a body, which is repeated with changing variables like the process shown in the repetition section, sometimes it may be better to create a shape and divide it into pieces to achieve a similar effect. By using traditional modeling techniques, base forms such as a surface or solid can be more easily defined, and then that simple form can be input into an algorithmic modeler where it is then divided along parametric lines. These parametric divisions can then become the basis of other variations of the form.

In this example, a single lofted surface cylinder shown in Figure 38 was created in Fusion360 and then imported into Dynamo.

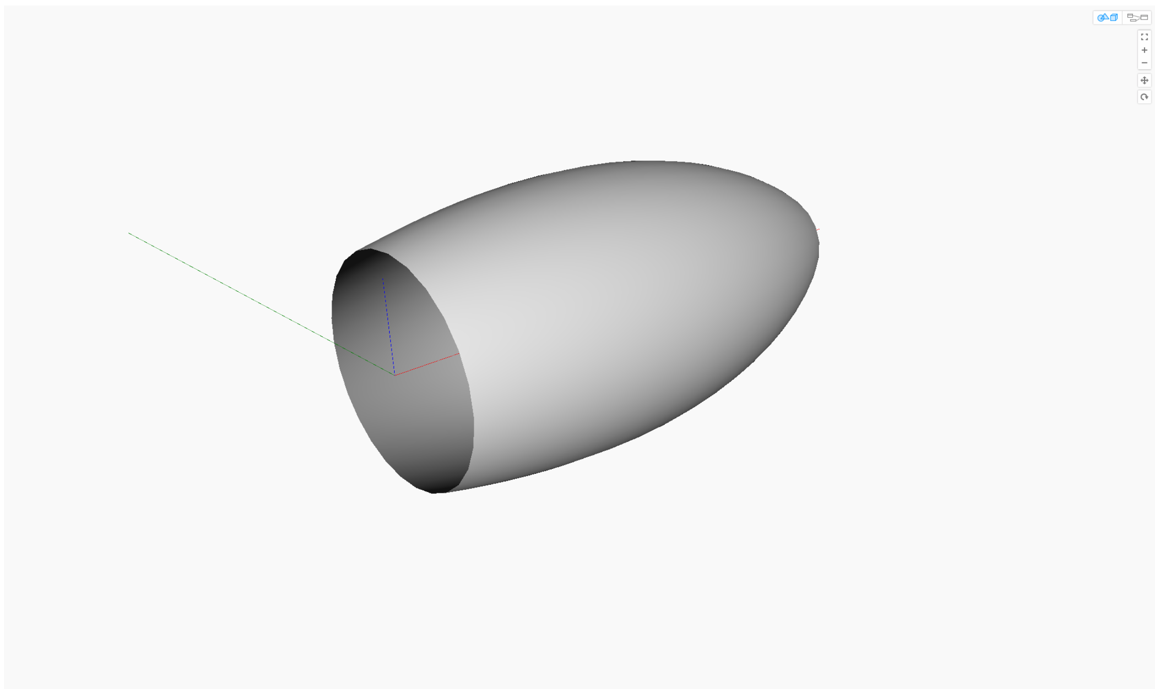


Figure 38: Curved cylinder surface imported from Fusion 360 to Dynamo

Once the cylinder is in place, isolines must be created along its surface in one direction. There exist simple nodes, which only need a surface input, parameter list inputs, and a selection of either the u or v direction with a switch. To create the list of parameters which must be greater than or equal to zero and less than or equal to 1, a basic list from 0 to 10 is created, then divided by ten, yielding: $[0, .1, .2, .3, .4, .5, .6, .7, .8, .9, 1]$. These parameter points are input into the node and provide the isolines at the different positions along the length of the surface seen in Figure 39.

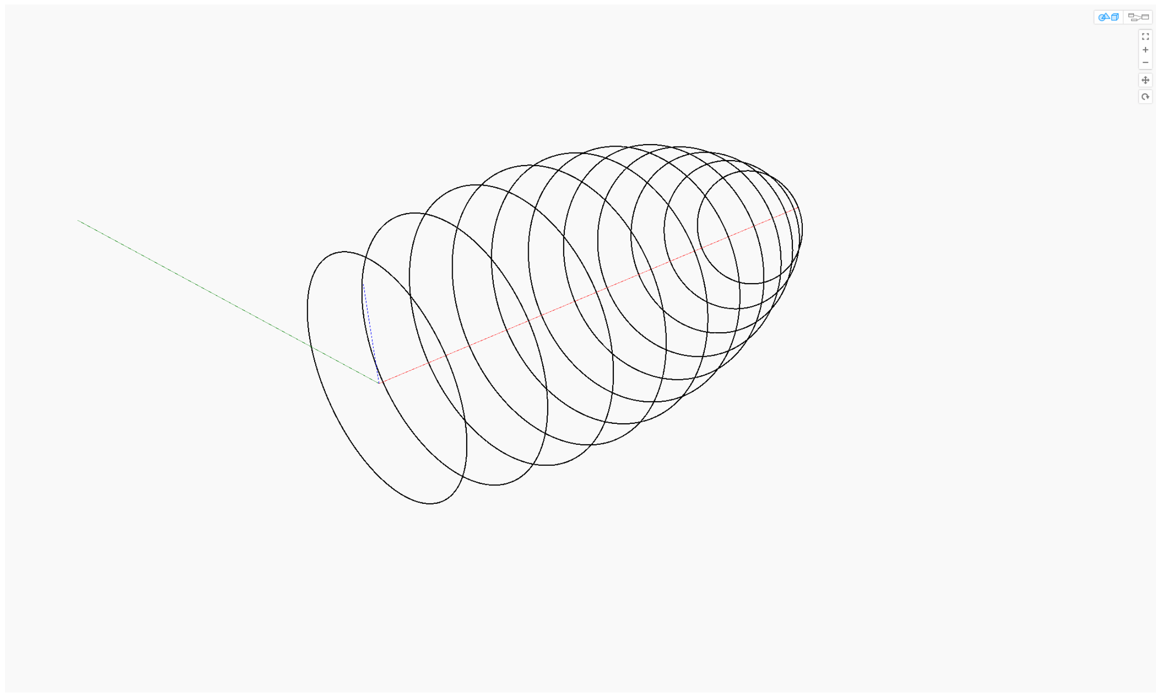


Figure 39: Isolines from parameter 0 to 1 in steps of .1 on an imported surface.

Now that the surface has been successfully "Divided" into equal parts, these isolines may be manipulated into something different. Since a cylindrical surface was used, the isolines in this direction are essentially circles. Every other circle around the central axis can now be scaled down and a loft between them is created to form the surface shown in Figure 40.

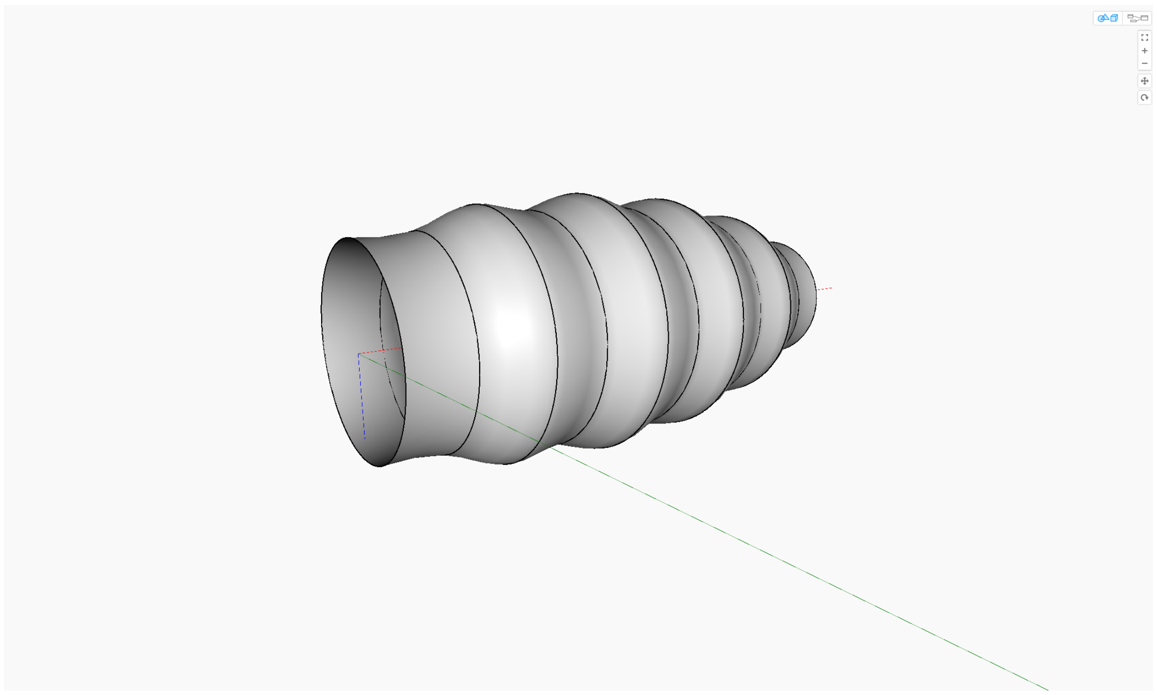


Figure 40: New lofted surface with alternating radii widths.

Once again, this may seem like a simple model, but one can first see how the system operates at a basic level, then see how changing one of the variables develops something which has a wholly different visual effect and structure. In this case, the number of isolines will be increased, which divide up the original surface and will increase the scaling for a more dramatic look. The surface can even be replaced with one which is shaped differently, but using the same code with different variables, the entire design takes on a different aesthetic. The result is shown in Figure 41.

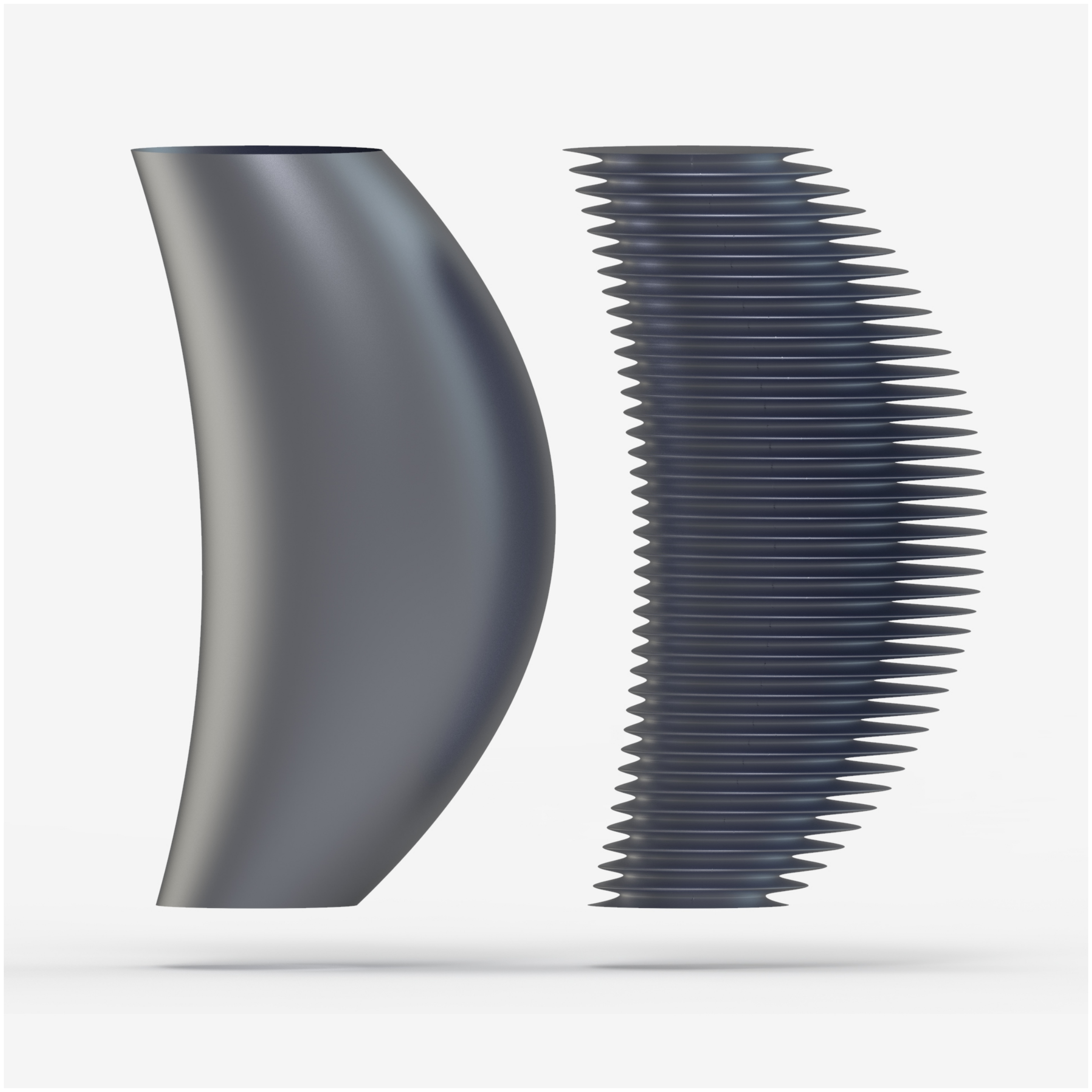


Figure 41: Left: New input surface shape. Right: New algorithm output with larger scaling and more isolines along its length.

By using a division of a pre-made surface like this, drastic changes can be made to a design input from a standard modeler. This can also save the designer time by letting them use tools they may already be more familiar with to create the geometries which will be imported into the algorithmic modeler.

Attractors

One of the benefits of working with data is that certain data types can be translated into other data types, easily. A common way this is done is by using an attractor system, where the distance from an object or series of objects can inform the shape or nature of the geometries. For this next example, data will be translated from distance to radii through the use of a remapping function.

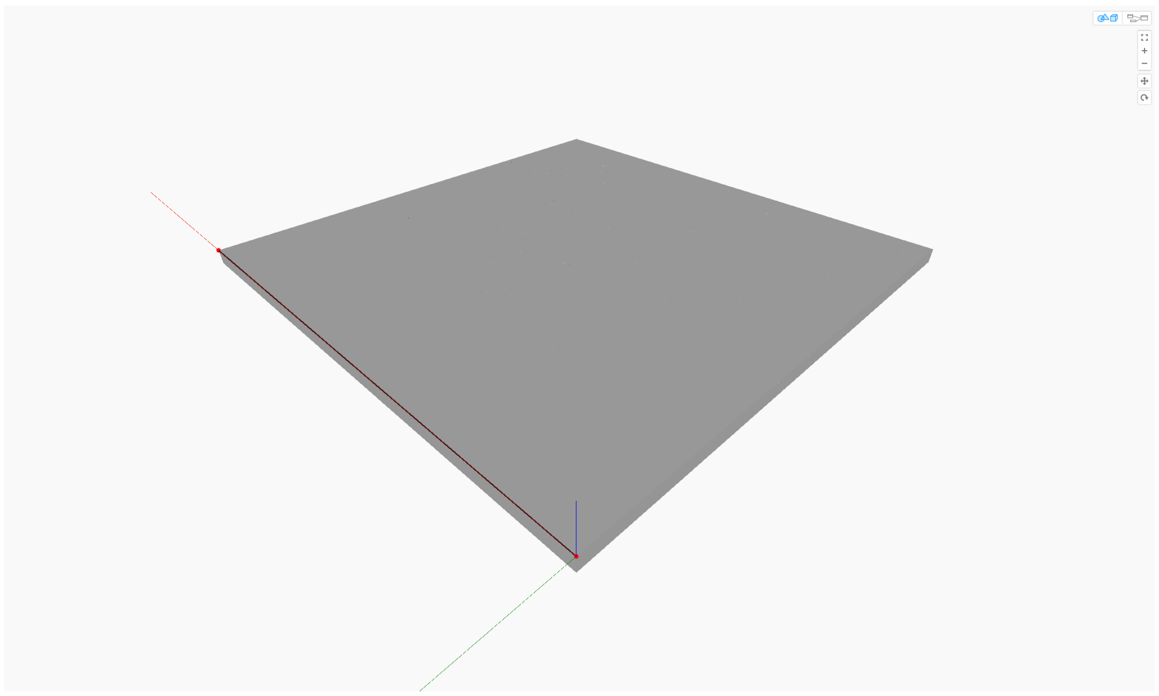


Figure 42: Large flat solid to use as a base surface.

This example will start with a flat base with points placed in a pattern along its surface. This was done using a grid-dividing node but can be done in many other ways as well. The surface with the system of points on top can be seen in Figure 43.

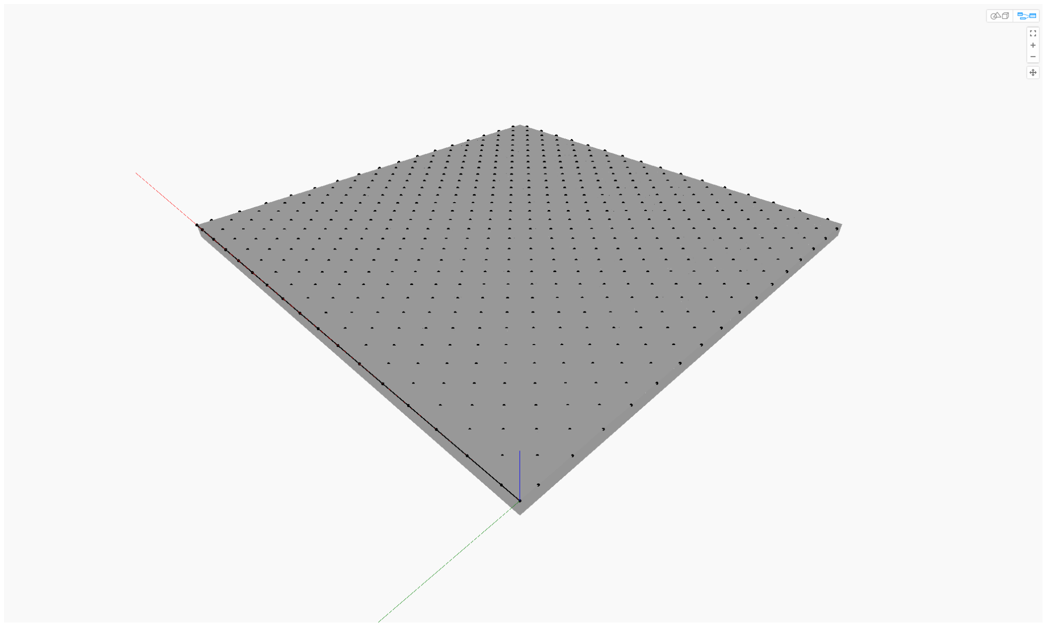


Figure 43: Surface with points in pattern

These points will serve as the location for a set of spheres shown in Figure 44. These spheres will initially be set to a constant radius.

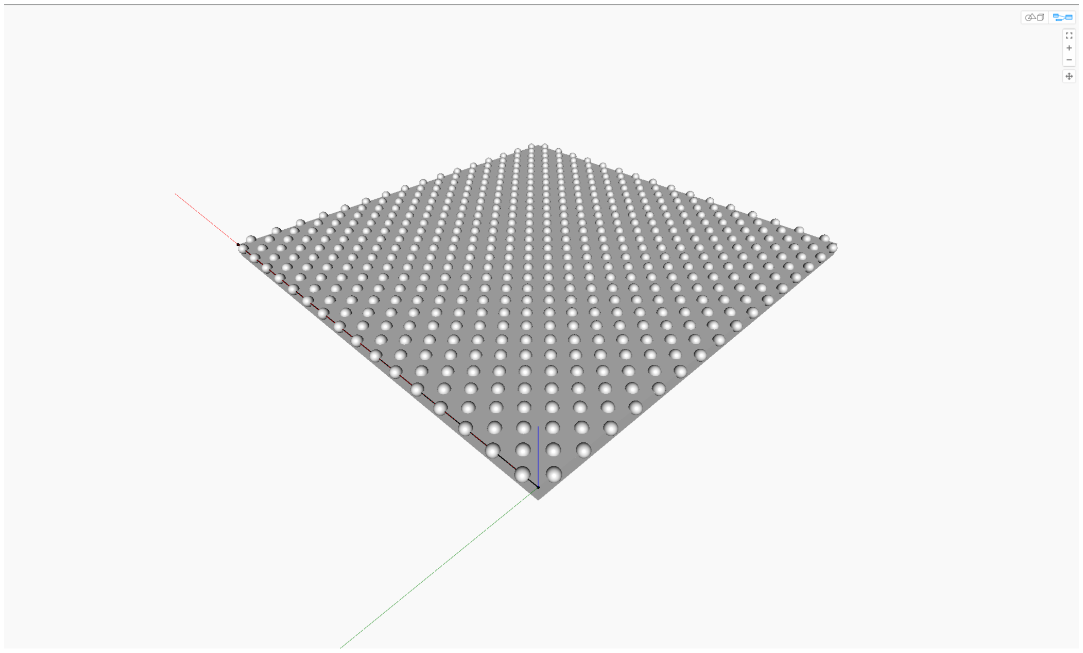


Figure 44: Grid of spheres of constant size on a surface

Now a remapping node will be used to change the size of the spheres. The remapping node will take in one piece of information such the distance from a sphere to the left bottom edge of the surface seen in this example, and use that range of numbers to define a different range of radii. Simply put, the remapping tool translates the distance from the spheres to the edge of the surface to their corresponding radii. Spheres closer to the edge of the surface are smaller, while spheres furthest from the edge are bigger.

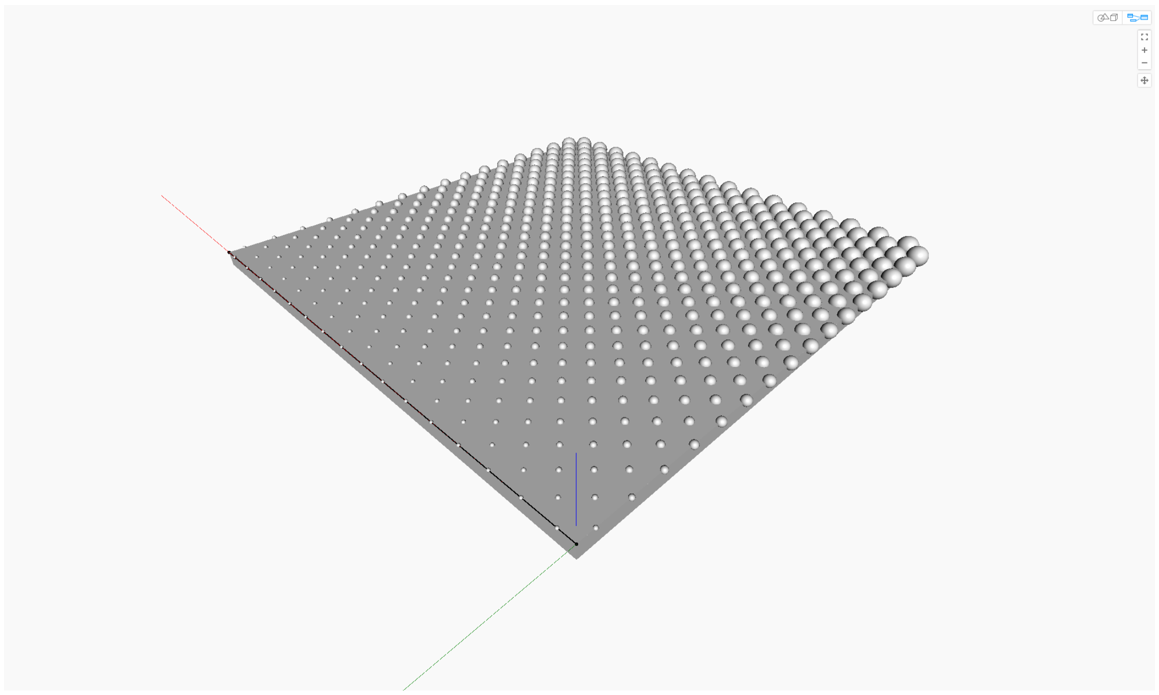


Figure 45: Remapped spheres

By using attractor systems like these, models can be created with great degrees of variability in the design without having to painstakingly define information in the system. For this example, the original geometry is cut by using the spheres to create a pocketed surface. This new geometry can then be patterned for more variation for use in paneling or creating some visual interest in a product.

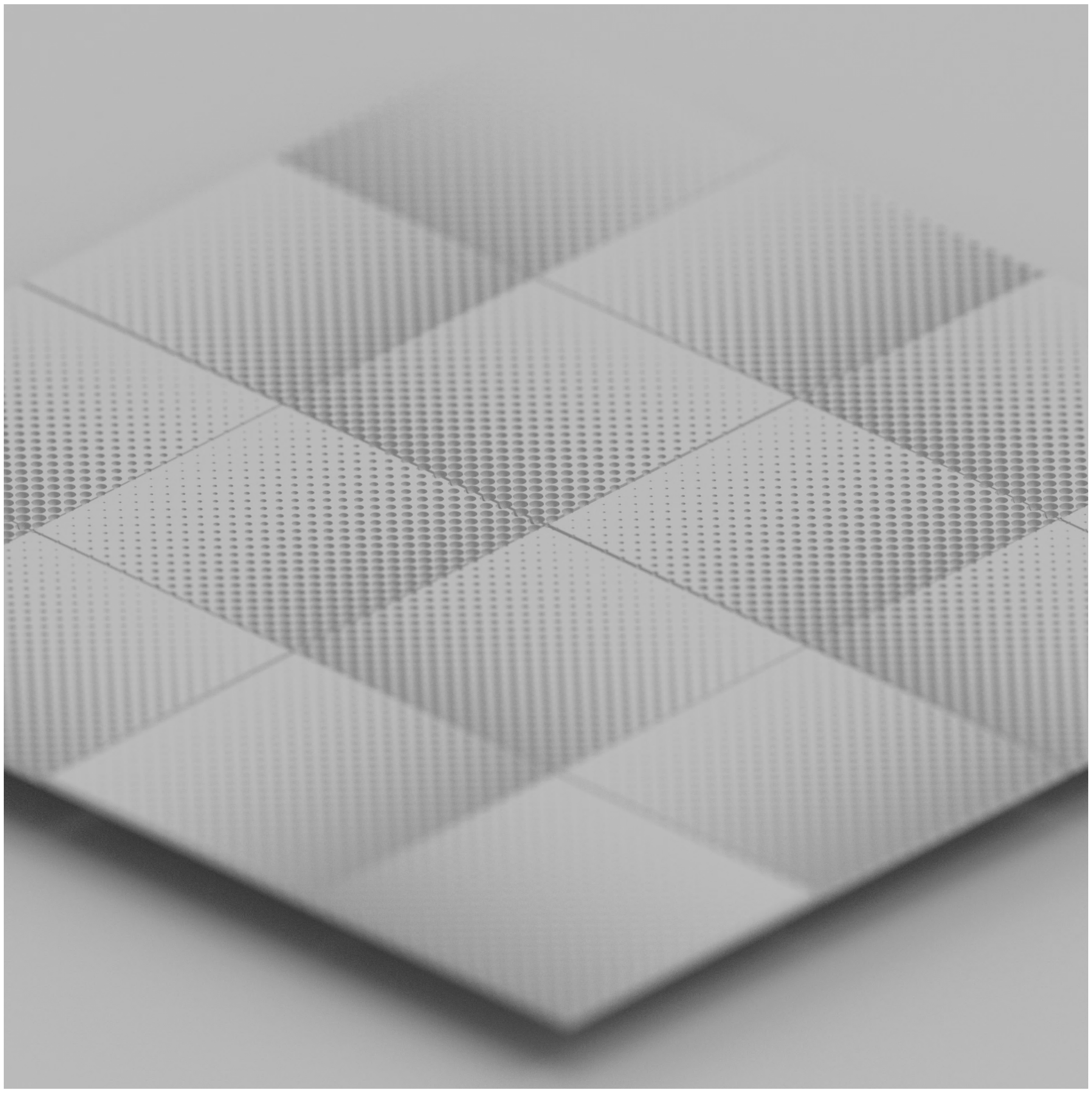


Figure 46: Rendered attractor panels

Surface Grids

It has been shown how to divide a surface into smaller sections, but sometimes a more complicated division of the surface may be needed. Luckily, nodes and components that are in algorithmic VPLs, or can be easily downloaded, can divide up a surface in various ways. These nodes usually take in a surface and apply a pattern such as a square, hexagonal, braced structure, or triangular grid. The outputs of these nodes include the lines and points that make up this new grid pattern. For this example, the basic tube surface used in previous examples will be used, shown in Figure 47.

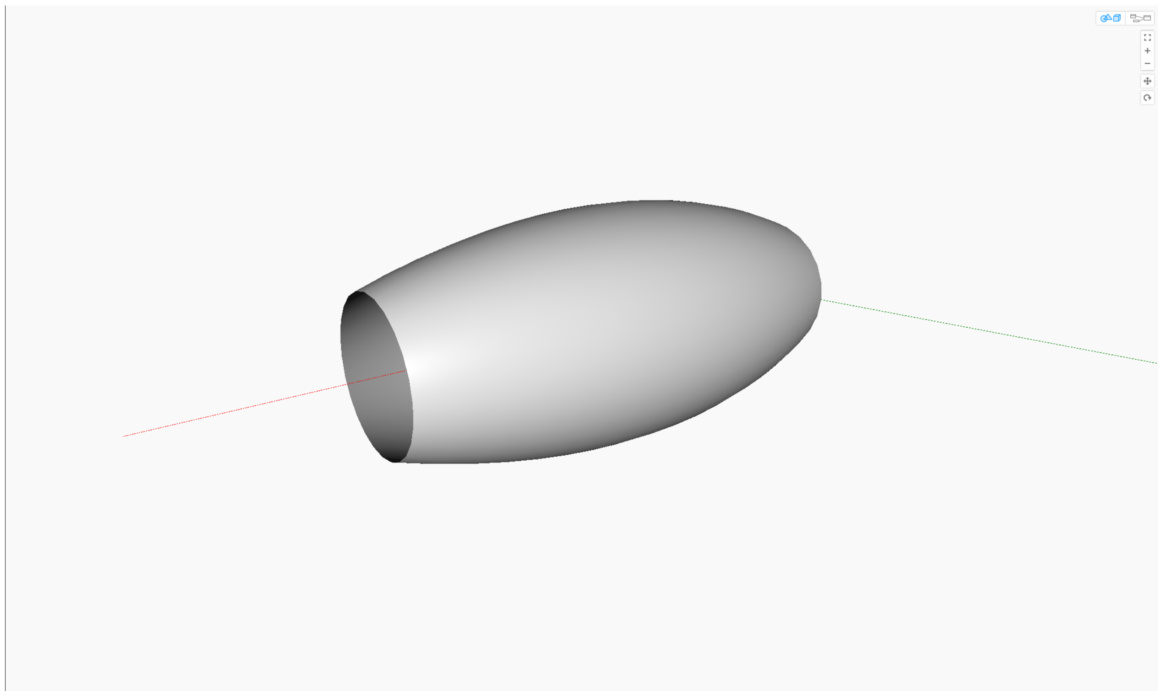


Figure 47: Basic single tube surface

This surface will be used to apply a hexagonal grid structure. The hexagonal grid node requires an input of a surface as well as the number of u and v divisions to form the grid. It has a variation factor input, which alters how uniform the sides of the hexagon are. This variation factor will be set so that the node creates a sort of inverted hexagon. The completed grid lines can be seen in Figure 48.

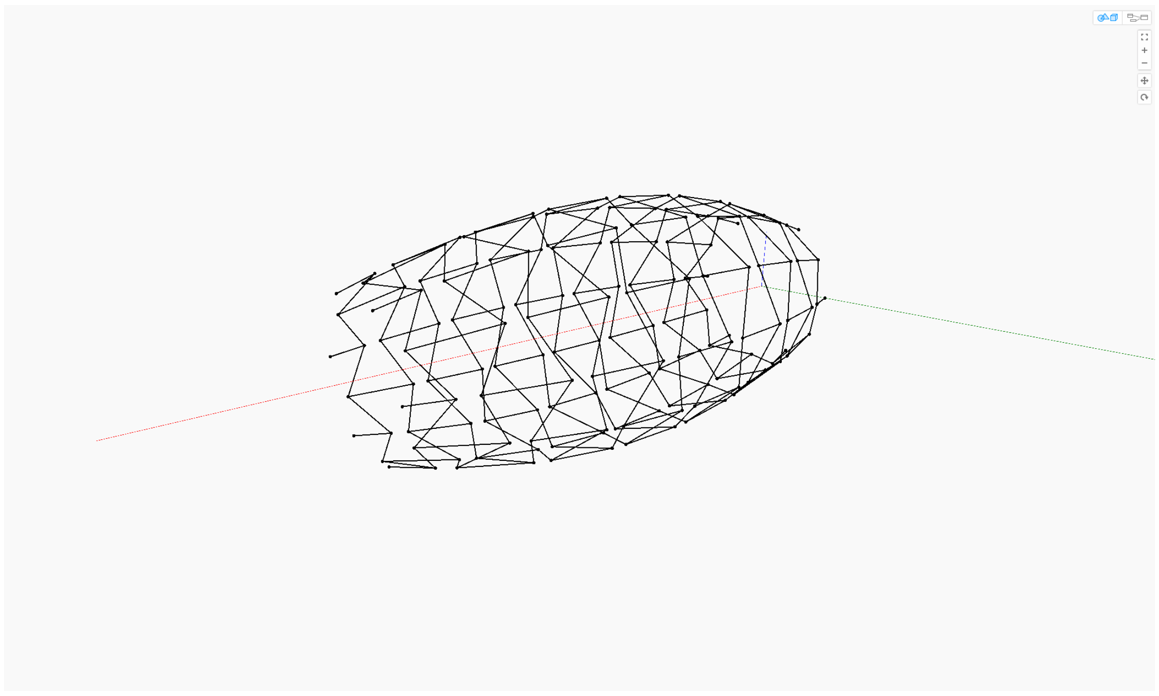


Figure 48: Surface hexagon grid

Now that the grid has been created, it can be used as the basis to form a solid geometry. A simple way of turning this Curve grid into a solid is to use a pipe node, which creates a pipe of a certain size around a curve. For this model, a T-Spline Surface node will be used, which takes in curves and creates a smooth T-Spline body around them, similar to a pipe node but with better transitions between the curves. This final body is shown in Figure 49. This method of using surfaces to define shapes, then creating a grid on top can help, creates different wire structures, which may be useful for making lightweight but strong parts in product designs.

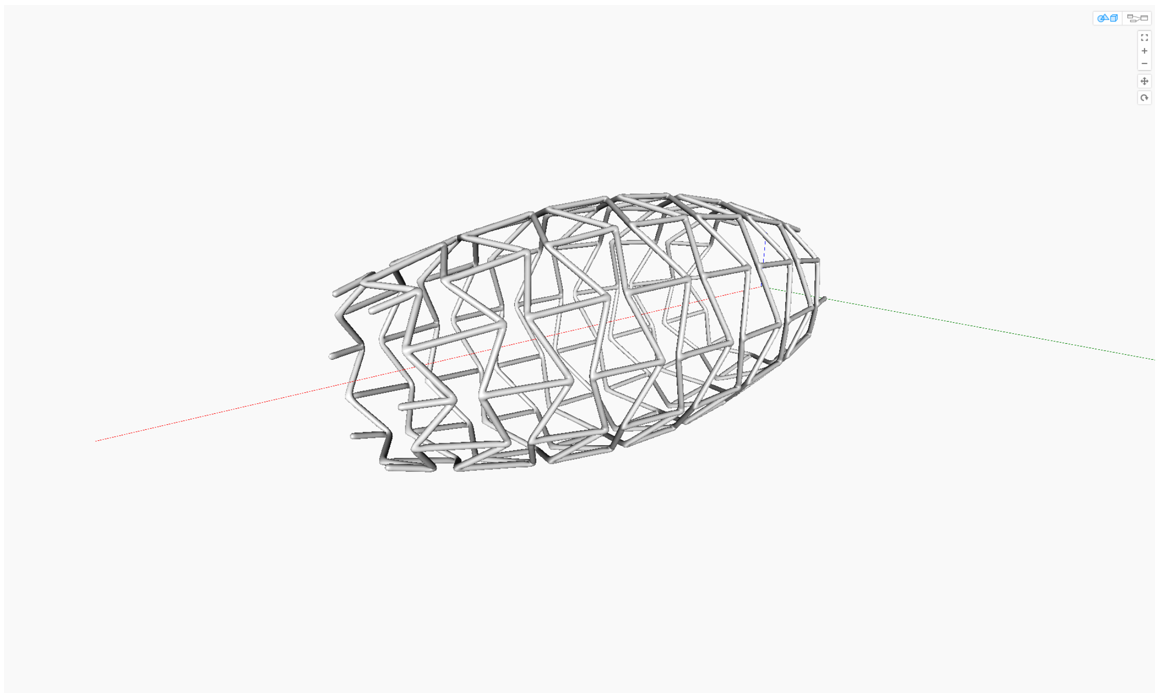


Figure 49: T-spline surface geometry of surface grid

In summary, the different algorithmic techniques shown in this document are just a sample of what can be done with algorithmic modeling. These few examples are meant to demonstrate how by using geometric terms and the available tools in VPLS, a variety of different designs and alterations can be created. Again, the full Dynamo code for each of the algorithmic systems shown in this chapter can be found at:
https://drive.google.com/drive/folders/1_iYBf6XisuOJ3Yhe0WXgLgz12W5Bn9bP?usp=sharing

Chapter 5: Application of Design Guidelines

This chapter will provide an in depth look at an example industrial design project: the development of a new phone case. This application project will work through the traditional industrial design and product development process, but also apply the design guidelines found in Chapter 3 as well as make use of the algorithmic terms, methods, and techniques found in Chapter 4. The purpose of this chapter is to show, specifically, how these guidelines are meant to be used, as well as provide an in-depth look at the algorithmic process including abstraction, successes and failures of the modeling process, and the final output.

Application Introduction

For this design application, a design prompt was chosen: Design a new smart phone case. The phone case should help keep the users from dropping their phone, but it should also consider aesthetics and marketability. This prompt was chosen because not only is it a very common product with a large number of users, but the users can be very diverse; almost everyone uses a cell phone. This variability will later help illustrate the power of algorithmic design. Before any algorithmic work, the process must start with the beginning of our guidelines and an evaluation of our goal and development of a physical solution. Additionally, it should be noted that for the sake of this project and the process it presents, the terms cell phone and smart phone are interchangeable and refer to the commonly produced mobile cellular device as well as the smart phone design standards which have been established at the time of writing (approximately 5” screen sizes, full touch screen interfaces, front & back cameras, cord charging systems, etc.) For reference, an archetype of the kind of phone to be designed for is the Apple iPhone X, released in 2017.

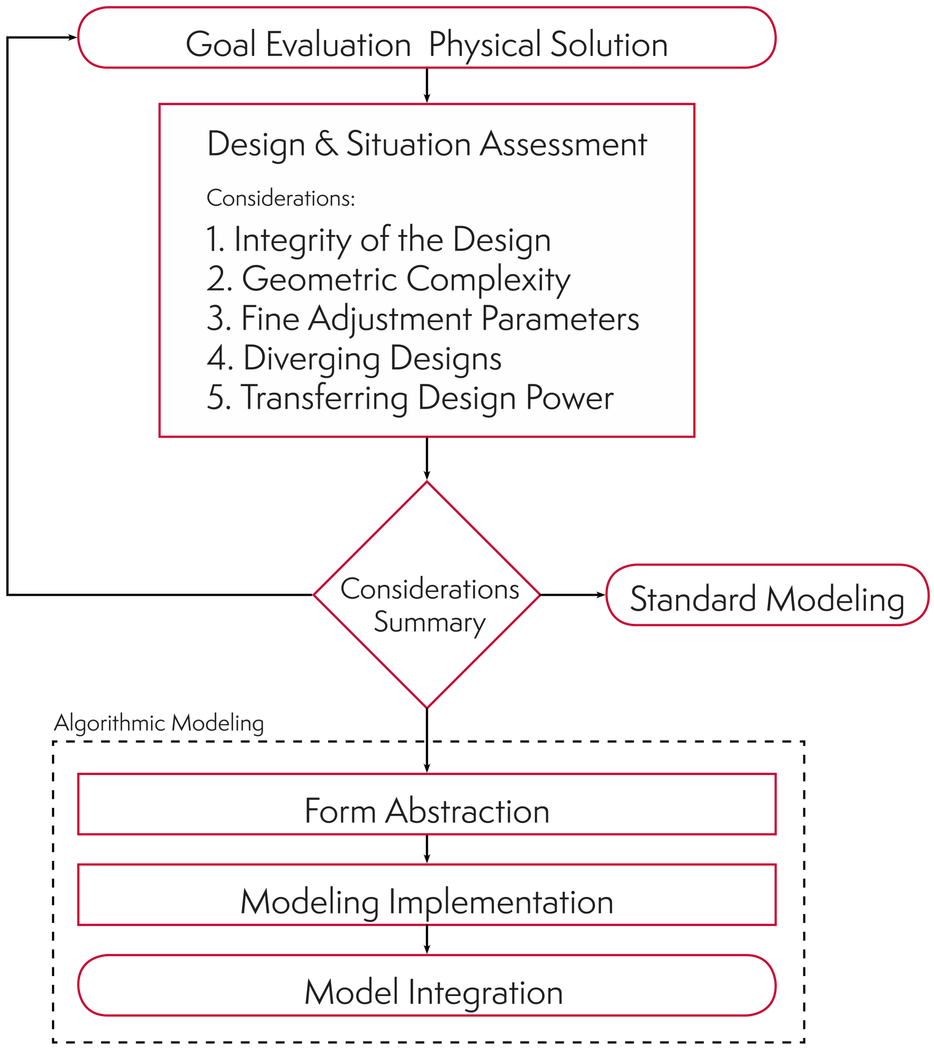


Figure 50 Algorithmic Design Guidelines

Application: Goal Evaluation & Physical Solution

This application will begin with the standard industrial design process and evaluation of the use of a cell phone. Starting with simple observations of a cell phone's use, interviews of people who use them, and hands-on testing, a few statements can be translated into goals for our solution.

- Cell phones are more commonly used with a single hand, rather than using both hands at the same time.
- There is preferred hand usage of the phone lasting when using it for more than a few seconds. This is usually the more dominant hand.
- Most phone drops happen during transitional movements e.g. removing from pockets, picking it up from a desk, moving phone in front of face, etc.
- During transitional movements, phones are held in a static place in the hand mostly though frictional forces and drops occur when the user doesn't apply enough force, then a "slip" occurs.
- The exact holding position of the phone varies from person to person, depending partially on their hand size as well as the size of the phone.
- Phone cases can act as a personal or fashion statement; their design and look play a major factor in their purchasing.

By using these statements found from the research, concept ideations must be made which could fulfill the new design goals: it must help prevent slipping, it must work for a wide range of people, and it must be aesthetically appealing. The concept sketch development around these goals can be seen in Figure 51.

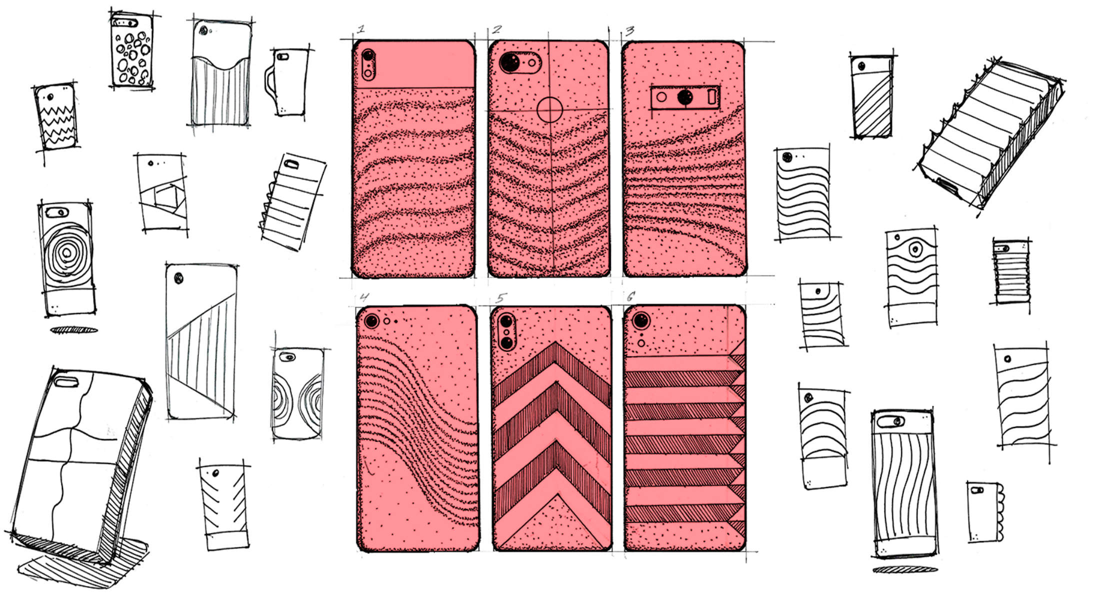


Figure 51: Concept sketch development of phone case

From this development phase, several final sketches were found to be appealing and matched the goals. For this project, the design shown in Figure 51 was chosen to be the final design to move ahead with. It provides an interesting geometry, which allows for additional grip on the back of the phone case.

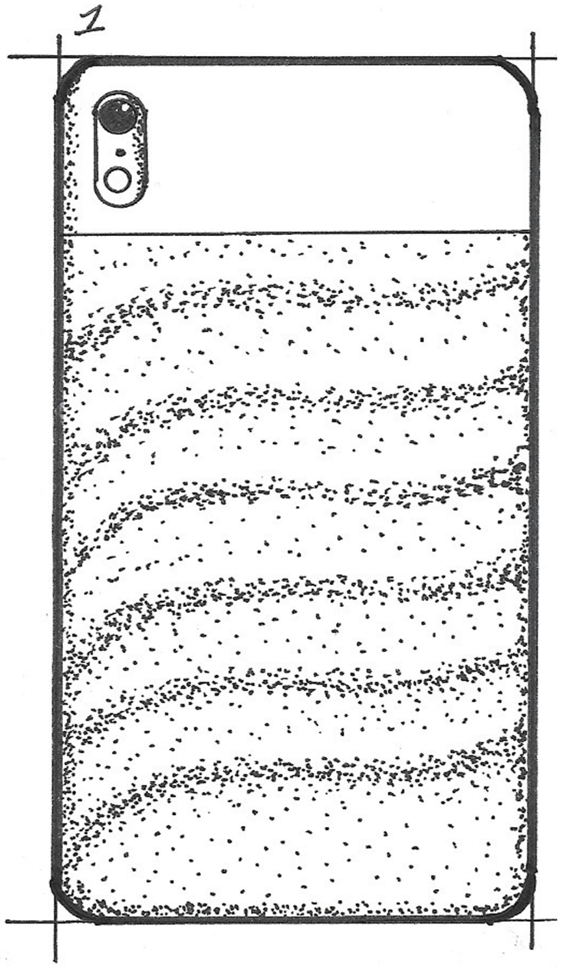


Figure 52: Finalized Phone Case Concept

Application: Design & Situation Assessment

The situation assessment will take into account not only the specific concept that has been chosen but the factors surrounding the project, as well. For the purposes of this project, it will be assumed that the designer has a developed knowledge of algorithmic systems and that, generally, being able to design faster is an advantage over working slower. The following Considerations will help determine whether the designer should proceed into algorithmic modeling.

Application Consideration 1: Design Integrity

As previously stated, much of the form is already defined due to it being linked to the size of the phone which the final phone case will be associated with. The rest of the form has already been developed through a series of iterations of sketches. If algorithmic modeling is to be used, there is a specific portion of the back which will be integrated into the algorithmic process, since the charging port and camera areas need to be untouched. The ridges will come off the surface of the case a small amount and then return to the base. What is not known though, is whether or not the ridges will wrap around the sides of the phone case or if they will just appear on the back and fade away toward the sides. The decision in either direction will depend on the capabilities of the algorithmic modeling system used, but both options seem like acceptable routes at this time.

So, in summation, the design or design direction is fairly well completed and a solid vision of what the end-product will look like is formed. The completeness of the design is not an indication that algorithmic modeling necessarily must be used, but that the designer may find it difficult to use algorithmic design if it is not complete; it is a stop gate before continuing to the rest of the considerations. Since the integrity of this design is fairly strong, the process will proceed.

Application Consideration 2: Geometric Complexity

The chosen concept has a fairly simple form, overall. The size of the product is determined by the size of the phone which it is associated with. The placement of the holes for cameras, buttons, and charging ports are also defined by the specific phone chosen. The most interesting parts are the wave shaped ridges which flow along the back of the case. Now, these ridges could be somewhat easily created with a series of defined curves in a standard modeler like Rhino3D. A curve network could loft these curves together to create this back surface, so at face value, the geometric necessity of an algorithmic modeler isn't readily apparent. If it is assumed that the number of ridges could potentially be changed to a much larger amount though, then the inherent geometric complexity increases. The amount of modeling time to create or edit the curves which define the ridges is directly related to the number of ridges in the system, which is still uncertain. So, in summation it can be said that the form is not necessarily complex, but for some situations it may be. If this were the only consideration to think about, a recommendation against algorithmic modeling could be made, but this is not the case.

Application Consideration 3: Fine Adjustment Parameters

There are several variables that could potentially be adjusted in this design. Since this project is already designed for different phone sizes, there is some built in variability, but the ridges on the back of the phone case have multiple parameters. Where the ridges start, where they end, how they curve, and how far off the surface the ridges move are all small adjustments which could be built to be manipulated. The need for customization based on variability between people's bodies as well as variability in their aesthetic choices is high for this particular project, and each design should reflect the needs of the end user. It may be the case that limits on these adjustments must be made so that the whole system works cohesively, but generally this project contains many variables which can be quickly fined-tuned in order to perfectly match the desired design of the end user. This consideration is one of the most important aspects of this particular design; the ability to make quick design changes almost necessitates the use of algorithmic design on its own.

Application Consideration 4: Diverging Designs

If the model were built in such a way where the portion of the phone case on which the ridges were made were isolated as part of the modeling process before the ridges were modeled, then that isolated portion could be used for any number of other design directions. Instead, it may be decided to use this portion to create a hexagonal cutout section, or a wireframe piped section. The ability to take this project in a different direction is fairly easy to do, assuming the model is structured correctly, and the need for design variety while still maintaining a consistently designed base makes this a great project for further iterations upon the same model. The desire for variety is high for the users, and alternated future designs are likely. Therefore, the need and ability to implement diverging designs, later on, is high for this project.

Application Consideration 5: Transferring of Design Power

As previously stated, the personalization aspect of this design means it could potentially give the design power to the end user so that they may define how their phone case will ultimately look. The phone brand, size, and the sizes and locations of the ridges are all variables which could be changed with no design or modeling experience, assuming the system is built properly. In this case, allowing the user to control exactly what they're getting could provide a very positive purchasing experience. Additionally, if someone else, or another designer, were to want to try to take the phone design in a different direction and add in diverging designs or other changes, it could be a somewhat easy experience for them as well. Overall, the ability to let the end user or others take control of their design is desirable in this situation.

Application Considerations Summary

After reviewing each of the considerations and assessing the design situation through each lens, it can be said that this particular design situation and concept is a conceptually complete design, is geometrically complex in certain aspects, will require many fine adjustment parameters, would benefit from having diverging design possibilities later on, and would benefit from giving the end users control over their particular design. Therefore, it can safely be said that this design situation calls for the use of algorithmic design for the modeling of this product.

Application Form Abstraction

Now that the decision to use algorithmic modeling has been made, there must be an attempt to define the parameters, geometry and methods used in the geometry through an abstraction before moving into the chosen VPL. This section will follow along the thought process of the algorithmic abstraction of our concept design. This will be presented as a series of drawings in order to help visualize the discrete steps, but in practice this is not entirely necessary.

The general logic of this design will be to algorithmically create a full phone case first, then use the created geometry to isolate the surfaces that the ridges will be added onto later. To begin, it can be recognized that the overall dimensions of the phone case will be dictated by the phone which is to be used. This algorithmic system will attempt to be usable for several different sizes of phones so the length, and width of the phone case will be adjustable parameters. These parameters can be defined as the size of a rectangular surface which will serve as the base, seen in Figure 53.

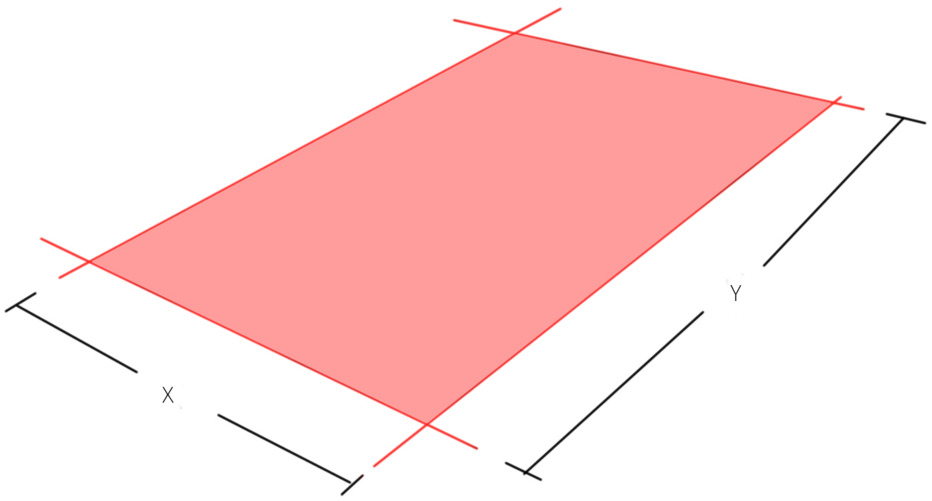


Figure 53: Base dimensions of the phone case as a surface.

Looking through the current phone market it’s evident that several brands have converged upon a similar design, which includes rounded corners on their phones. For an accurate footprint, radii will need to be added to the corners of the base surface, which will also be an adjustable parameter for different models, shown in Figure 54.

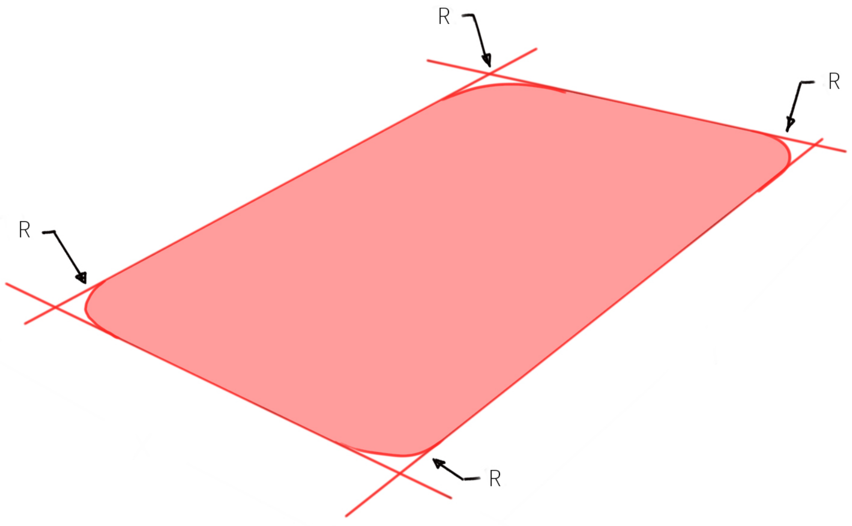


Figure 54: Base surface with added corner radii

So, with a footprint constructed, next the sides of the phone case are created. These are usually a near constant radius which curves around the sides of the phone. This surface is defined with a sweep along the outside of the established footprint with a profile of an arc with a variable size. This swept surface is shown in Figure 55 with a section view of the arc used to sweep the profile.

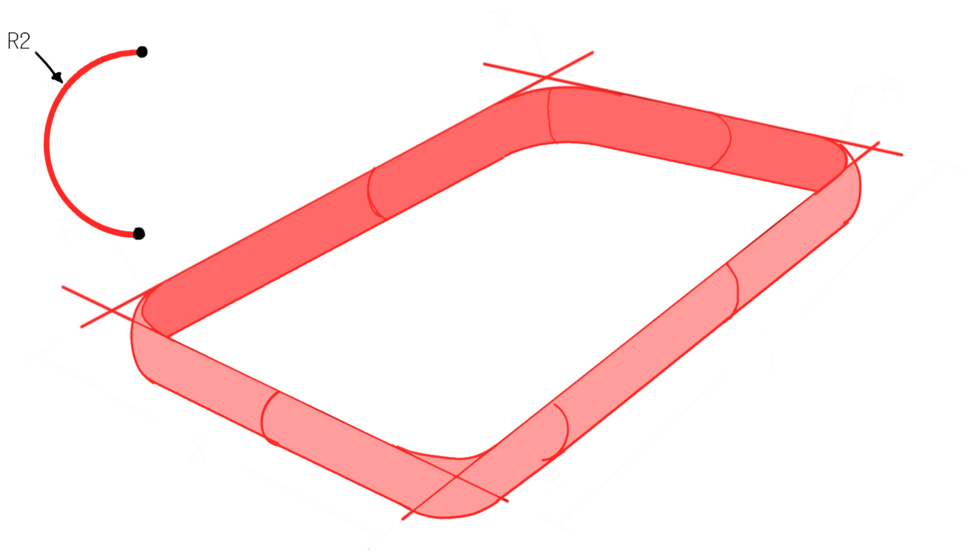


Figure 55: Swept surface using arc to define phone case sides.

Next, the sections of the surface that will be used for the ridges must be defined. The side surface must be split at variable distances from the edge of the case to form two new smaller surfaces, shown in Figure 56. The edges of these split pieces will later define the ridge workpiece.

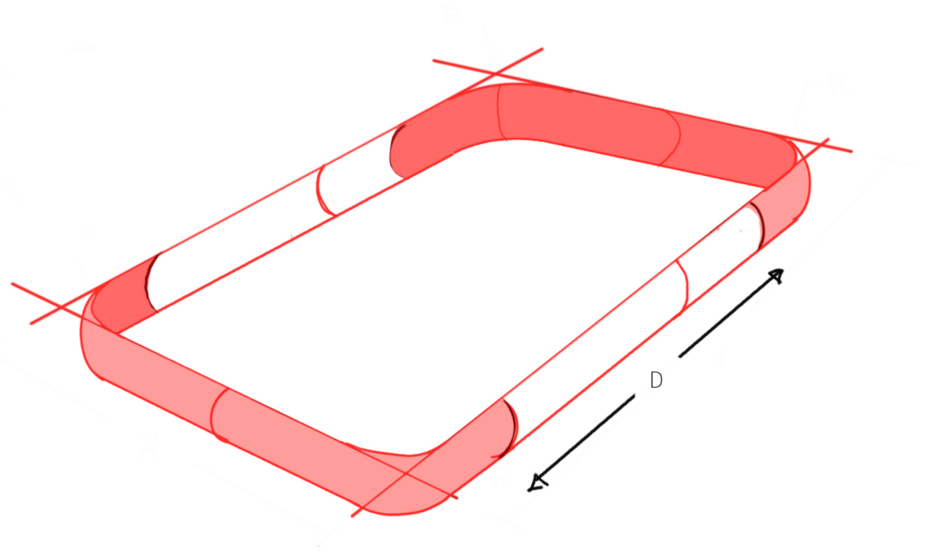


Figure 56: Phone case side surfaces split at a variable distance

With the two new surfaces created, now the curved edges can be used as a basis to form the new ridge surface workspace. It takes in the curved sides and creates lines from one surface to another to define this new surface, shown in Figure 57.

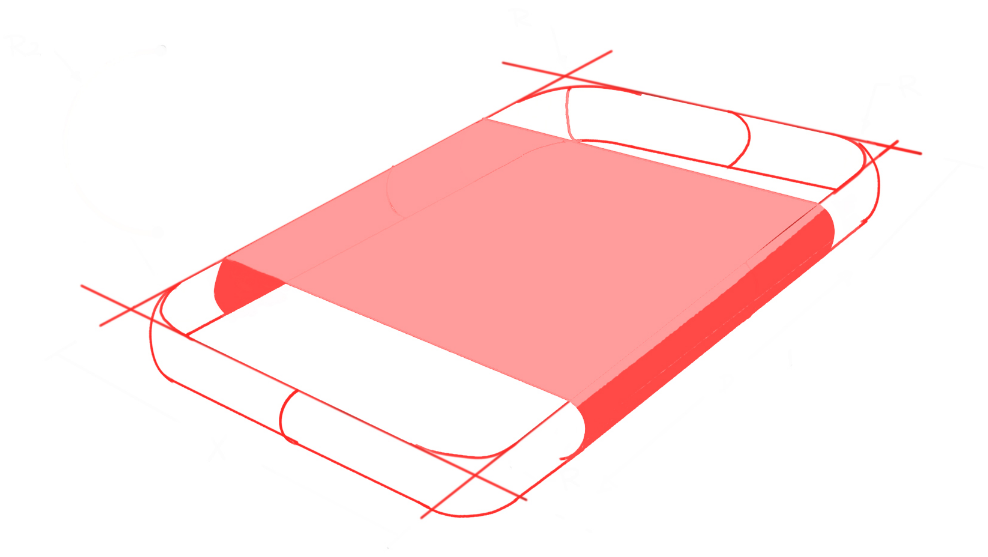


Figure 57: Full ridge workspace surface.

Now, it may be possible that this surface with the curves extending around the sides may be difficult to work with in the VPL, but this will be attempted anyways. If this surface does prove to be too unwieldy, it can be subdivided even further into a completely flat plane, shown in Figure 58.

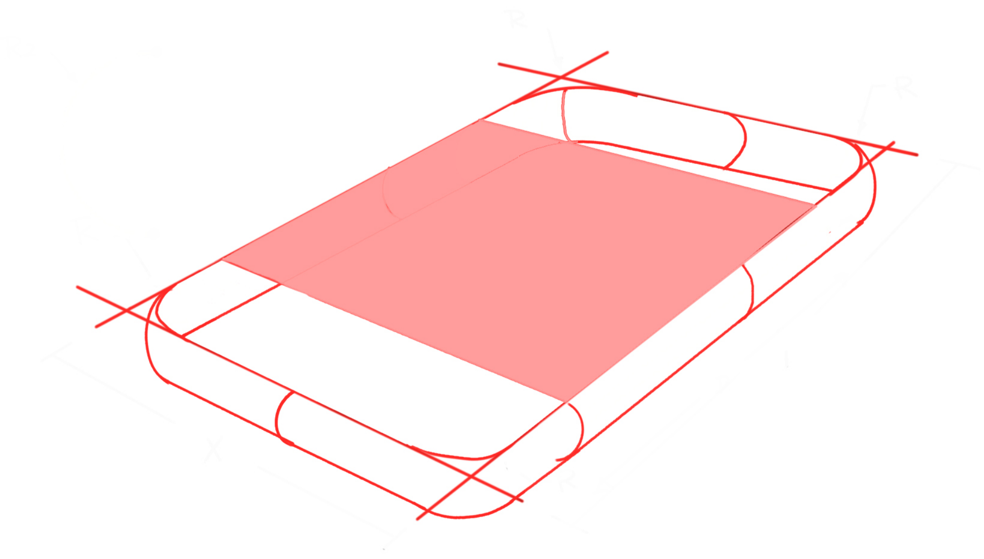


Figure 58: Further divided ridge work surface

Regardless of which method is chosen, the next steps will remain roughly the same. With the ridge work surface defined, the top view can be viewed to better understand it. The lines along the sides of the surface, and one in the middle of the surface, will be defined and shown in Figure 59. The start and end points of each of these lines will be variable but will likely need to have limits set on their locations. The sizes and locations of these lines will define the overall flow of the ridges along the back of the phone case. For example, if the lines are very short, and located only in the middle, then the ridges will occupy a narrow area in the middle of the surface. It is this variability that will actually allow us to create several of the finalized concepts from those that were developed earlier, since a few were all similar ridge systems, but with varying locations and ridge movements.

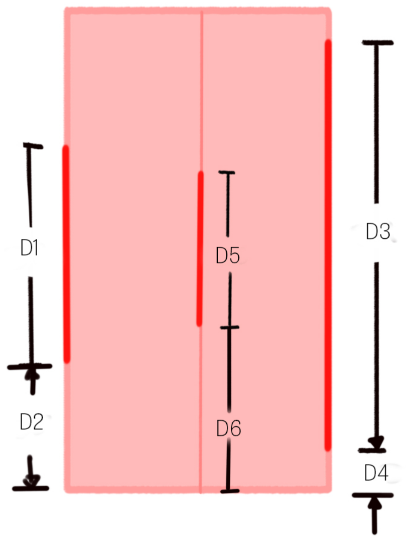


Figure 59: Ridge work surface with three variable lines defined

Now with the sizes of the lines defined, each line can be subdivided into a series of points along its length. The number of points which subdivide each line will be the basis for the number of ridges on the surface; the more points there are, the more ridges will be created. It is also important to make sure that the number of points is the same on each line since they will be connected with splines. Therefore, each point has a corresponding point on another line to connect to.

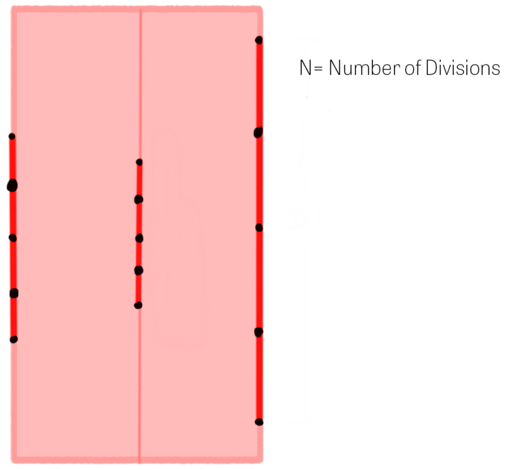


Figure 60: Surface with subdivided lines.

The next step is to take each of these points along the lines and connect them with a spline. The form of the ridges is starting to take shape, now. Each spline will define the center point of a ridge, but each spline may be a hill or a valley in this system.

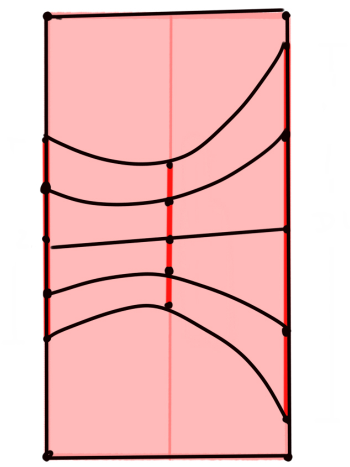


Figure 61: Splines connecting points along subdivided lines.

Now, these lines need to be moved into three dimensions. This is done by moving the points on the middle row up a variable amount. Every other point will be moved in order to create the ridges.

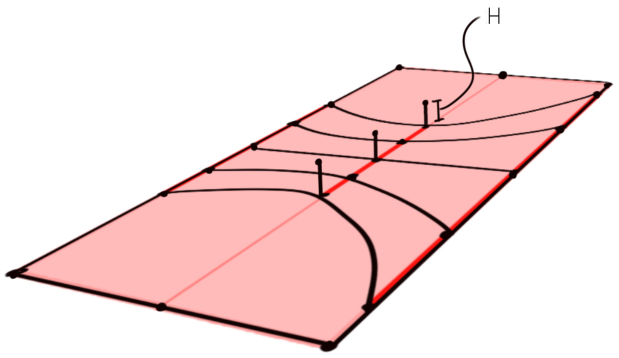


Figure 62: Surface with middle subdividing points moved vertically.

With these points now moved vertically, new splines can be defined that will pass through the points on the sides and the shifted middle points. It can be seen that although this method does form some ridges, more subdivided points are needed to achieve the desired wavy effect illustrated in the concept sketches.

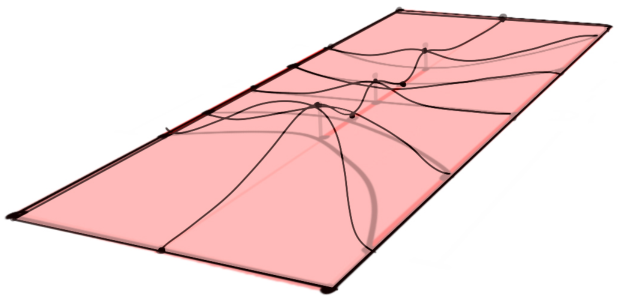


Figure 63: Splines running through the translated points

From the top view, shown in Figure 64, it's shown how each of the new splines, as well as the lines from the sides of the base surface, define a new network surface, creating the final ridged surface.

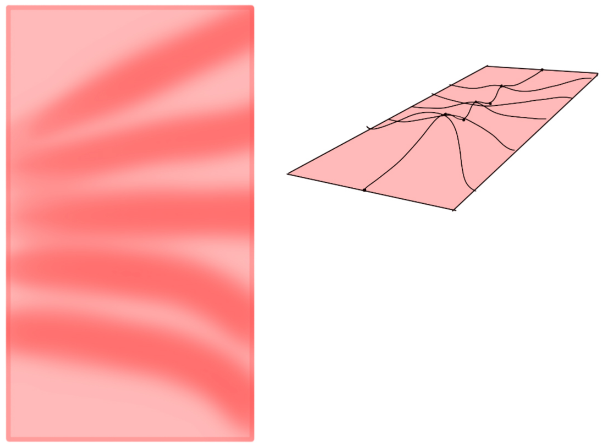


Figure 64: Final ridged surface piece

The only thing left to do is to take this surface piece and add back in any side surfaces which were hidden or removed in the building process. Cuts or holes must also be defined, which will be needed for any cameras, charging ports, or fingerprint scanners. These will be defined by simple Boolean subtractions from solids which are placed at predefined locations based on the type of phone selected. Figure 64 shows the final abstracted form of the phone case, with full sides and cutouts.

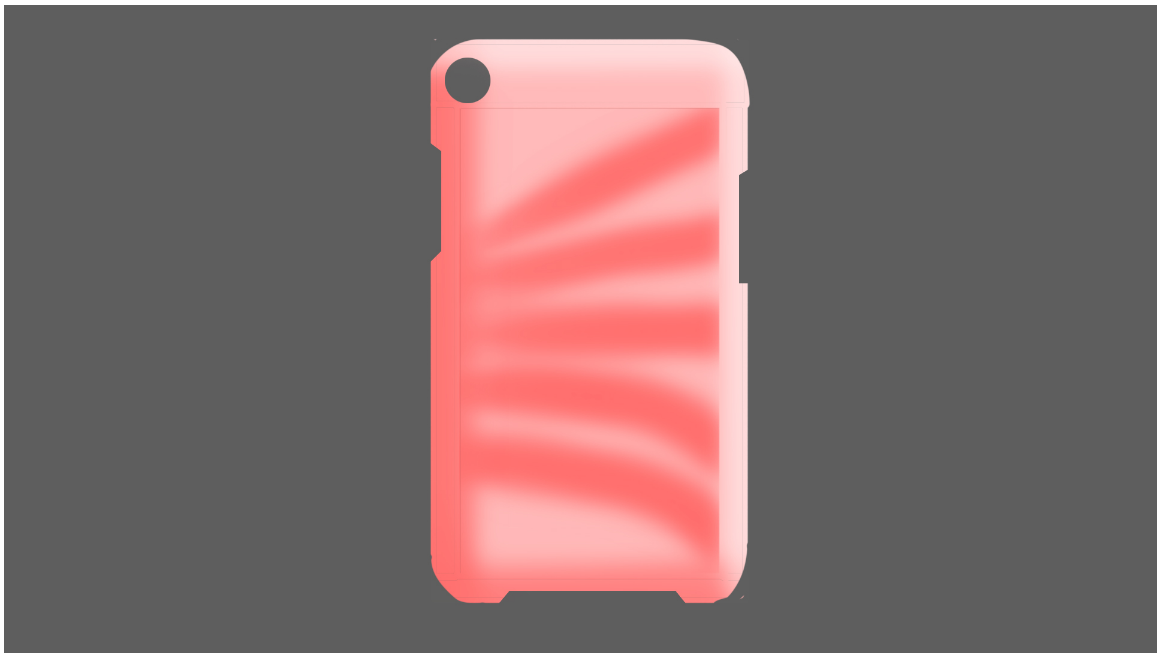


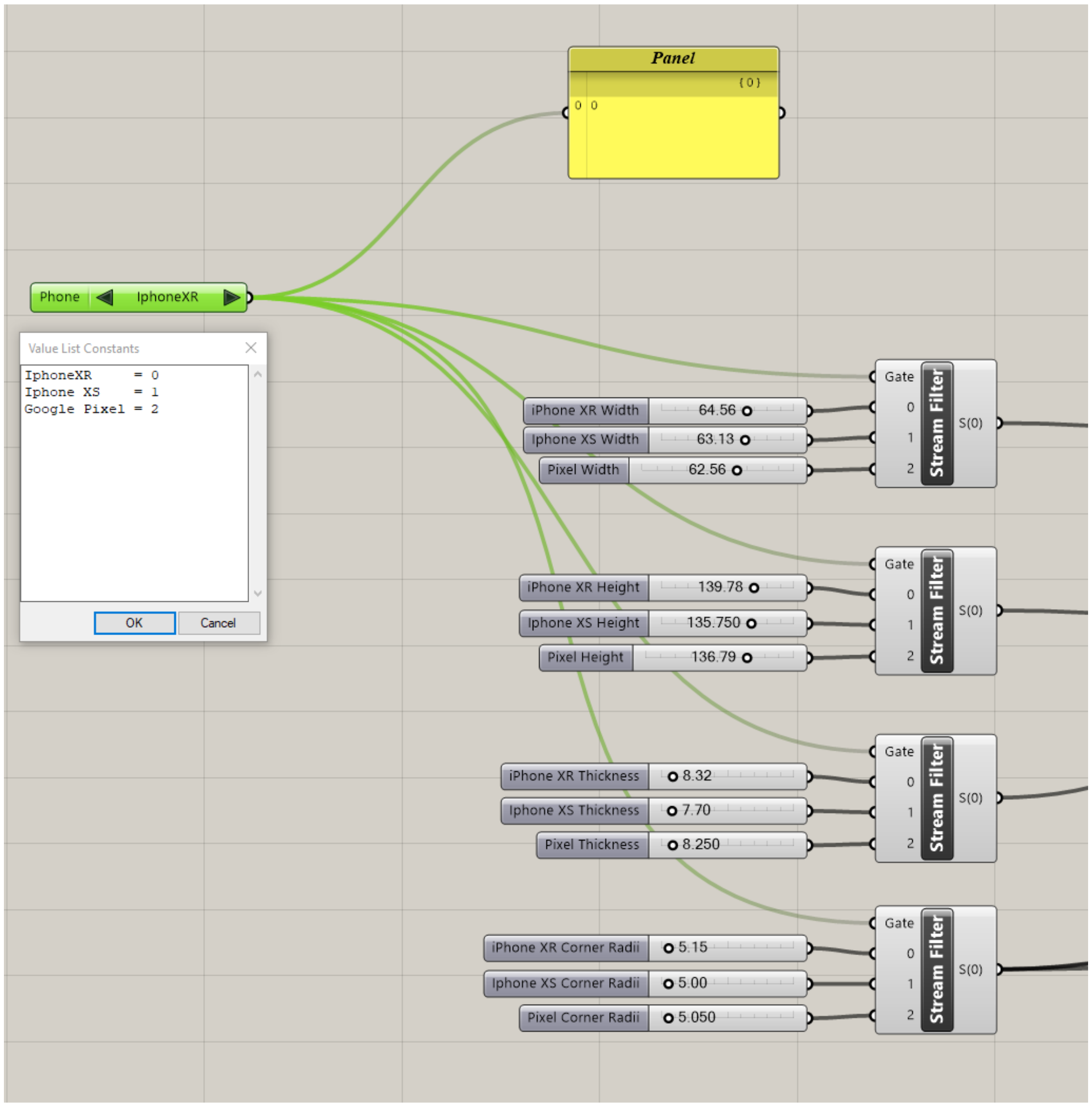
Figure 65: Finalized Phone Case Abstraction

Through this abstraction process one can see how attempting to define the geometry using mathematical and geometric terms allows the designer to test the validity of their design for algorithmic systems without having to fully jump into the modeling process. The variables and steps which will be needed for the modeling process can also be defined and planned. After working through this abstraction, work in the chosen modeling environment can begin.

Application Modeling Implementation

For the implementation of this algorithmic design, Grasshopper will be used as the chosen VPL. It is the more commonly used and written about software, so I believe that it is the better choice for a document such as this. This algorithmic nature of these systems means that the following construction is possible in other systems, but the minute details such as node or component names and small logical flows may be different.

To begin, since each design will be for a particular brand of phone, it may be beneficial to start with a phone brand "selector" which will choose from several defined brands. For this project, three phone types will be used: the Apple iPhone XR, Apple iPhone XS, and the Google Pixel 3. Given what is known from the abstraction process, a few key dimensions for each phone are measured: the active area height and width, the thickness, and the radii of the corners of the phone. Using this information, a selector system is created which activates the associated dimension, for use in the algorithm, through the use of several Stream Filter components, shown in Figure 66. A Stream Filter component produces a defined output from a list based on a selected input.



```
IphoneXR     = 0
Iphone XS    = 1
Google Pixel = 2
```

Figure 66: Phone Selection and Stream Filter systems

Once a phone is selected and the corresponding dimensions are activated, these dimensions are used to define the x and y coordinates of two points through a Construct Point component. A third point is defined at (0,0) for construction purposes. These points are then used as inputs for a Rectangle From 3 Points component. This creates a rectangular curve with one corner at the origin with dimensions equal to that of the selected phone brand.

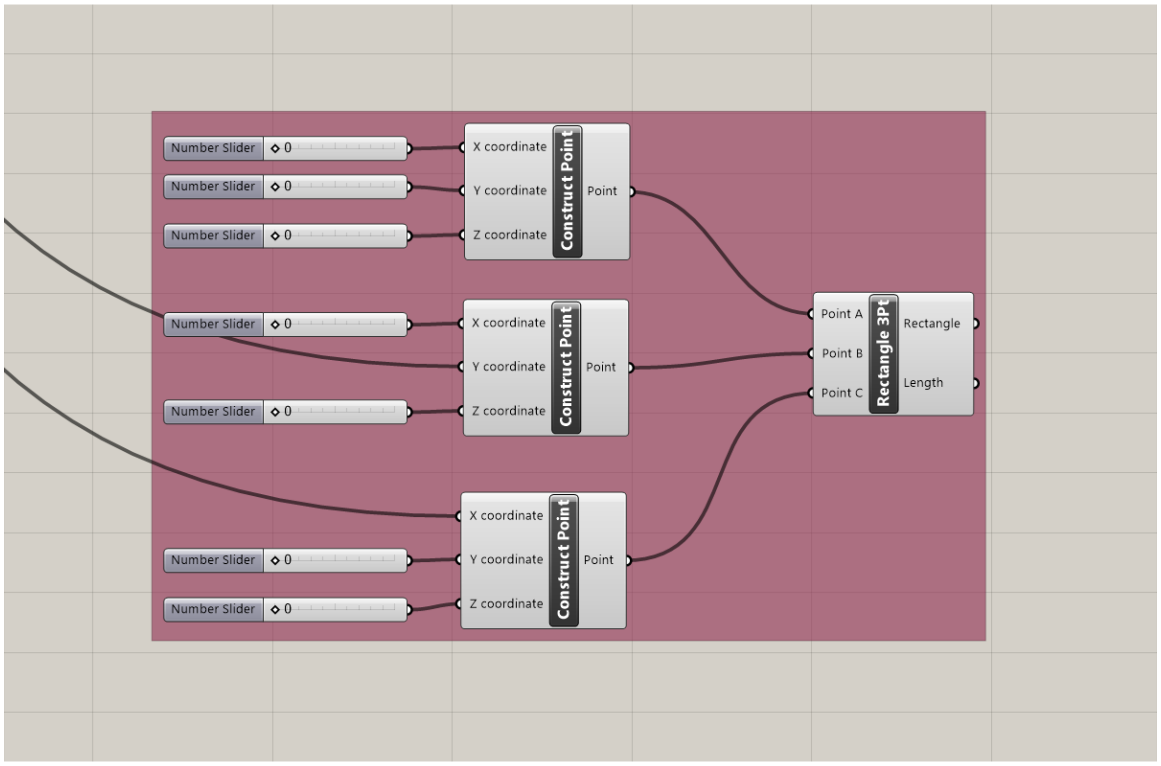


Figure 67: Construct Points and Rectangle From 3 Points components.

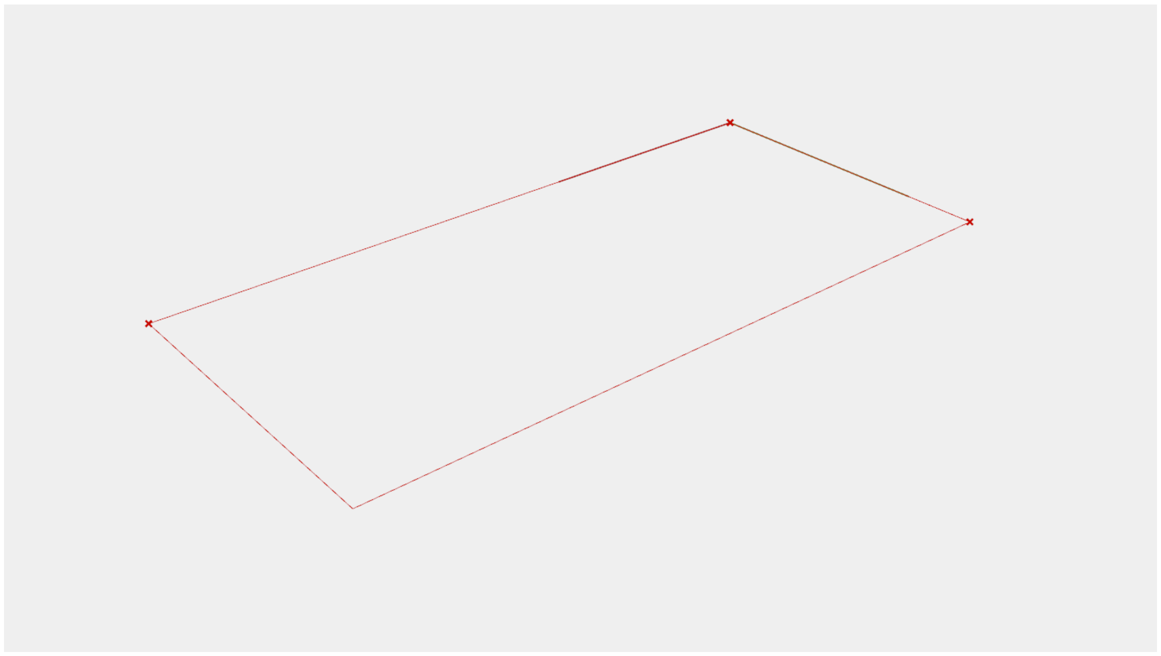


Figure 68: Rectangle Curve as a result of the Rectangle From 3 Points components.

Next, a curve which will define the round outside surface of the phone case must be made. To do this, a point is defined on the rectangular curve at a distance equal to the curvature of the phone's corners. A second point is defined at the same location, but at a negative Z distance equal to the thickness of the selected phone. These two points are used to define an Arc component, which is tangent to the rectangle curve and has a diameter equal to the phone thickness, shown in Figure 69. A Fit Curve component is then used to change the Arc output to a general Curve output.

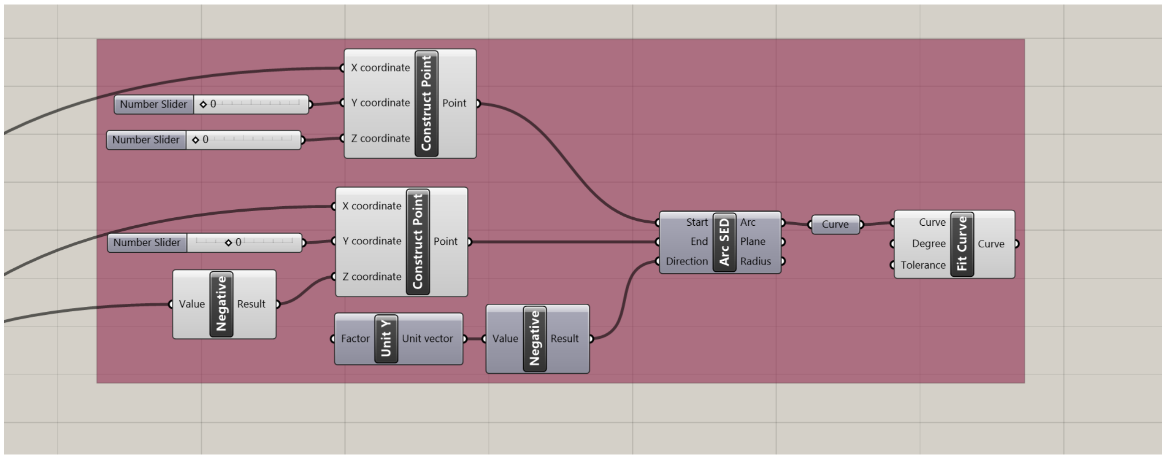


Figure 69: Components defining an arc

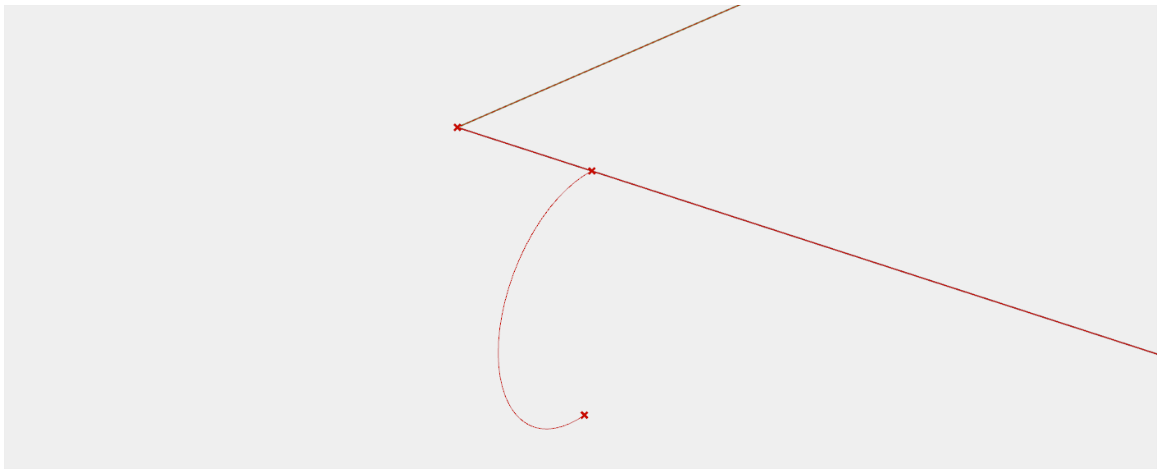


Figure 70: A defined curve the same thickness of the selected phone case

Next, the rectangular curve is used as an input for the Fillet component, which places radii on each of the corners equal to the phone corner radii selected. This new filleted curve is used with a Boundary Surface component to create a surface in the shape of this filleted curve, shown in Figure 71.

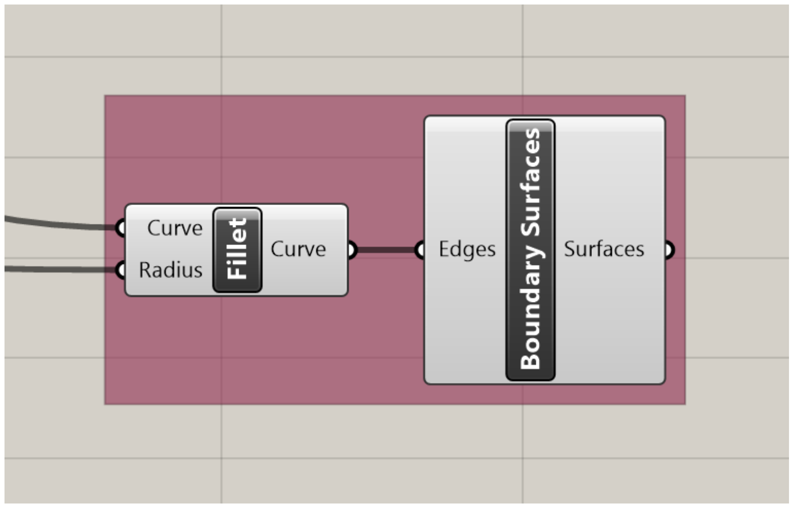


Figure 71: Fillet & Boundary Surface components.

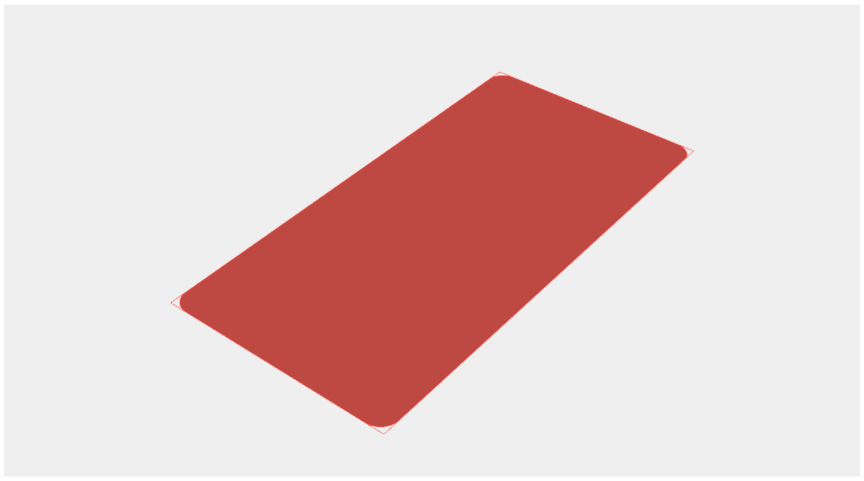


Figure 72: Surface of the filleted curve

This filleted surface created is a defined “back” of the phone case, but the other surfaces still must be added to form the initial geometry of the full phone case. Using both the arc shaped curve and the filleted curve created earlier, a Sweep 1 Rail component can be used to define the "side" of the phone case. The filleted curve is used as the rail, and the arc is used as the sections.

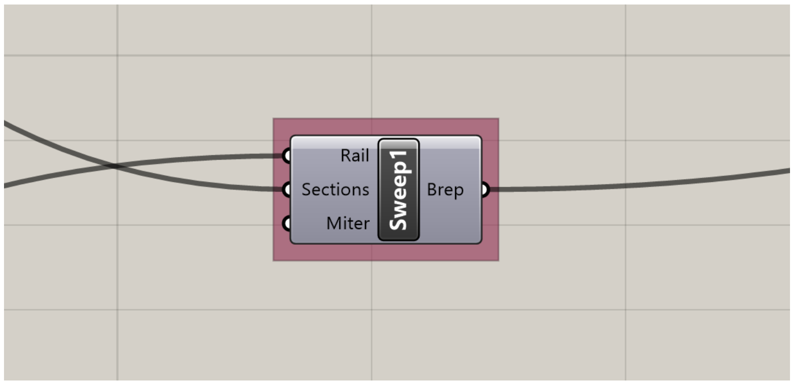


Figure 73: Sweep 1 Rail component

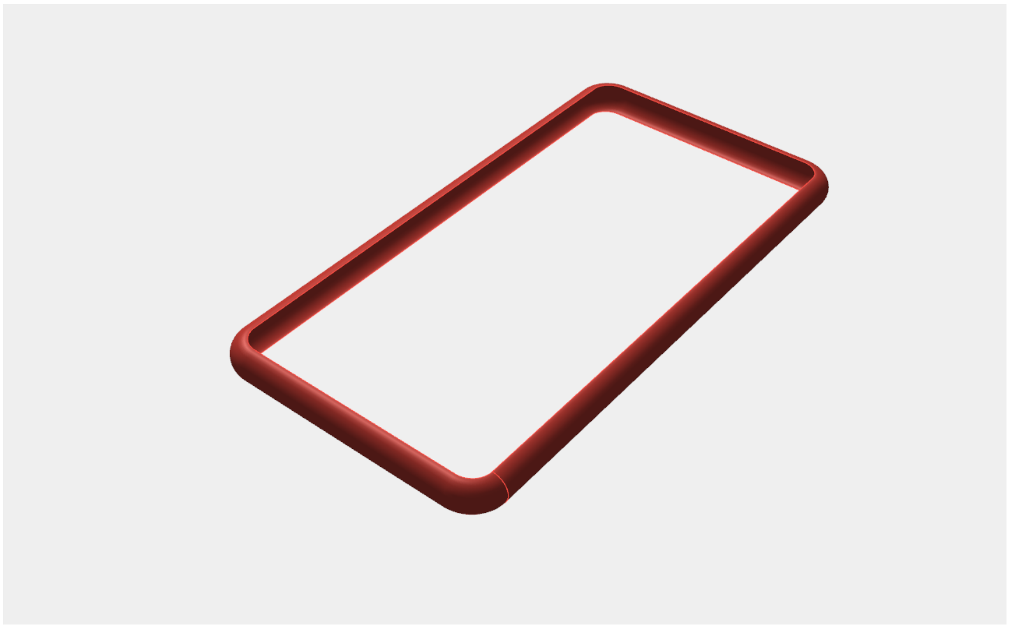


Figure 74: Sweep of arc around filleted curve.

In order to create the rounded work surface of a variable size on the back of the phone case, this sweep needs to be split at certain points in order to create edges to work from. To do this, an Iso Curve component is used to define two curves on the rectangular surface at specified distances along its length. These curves are then extended past the limits of the sweep using an Extend Curve component. These two curves are then extruded into a surface in the negative Z direction, so that the surfaces fully intersect with the swept surface. In order to make sure that the surfaces are the appropriate sizes, the Extend Curve distance and Extrude Curve distances are equal to the selected phone thickness. This ensures that the surfaces always fully intersect with the swept surface. The full components for the creation of these surfaces are shown in Figure 75.

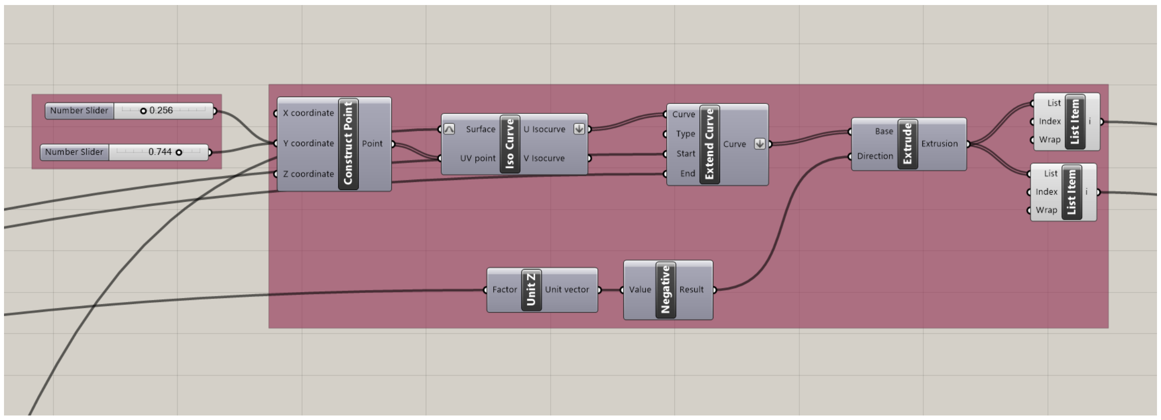


Figure 75: Components for defining extruded surfaces at specified distances along a line, which intersect with the swept surface.

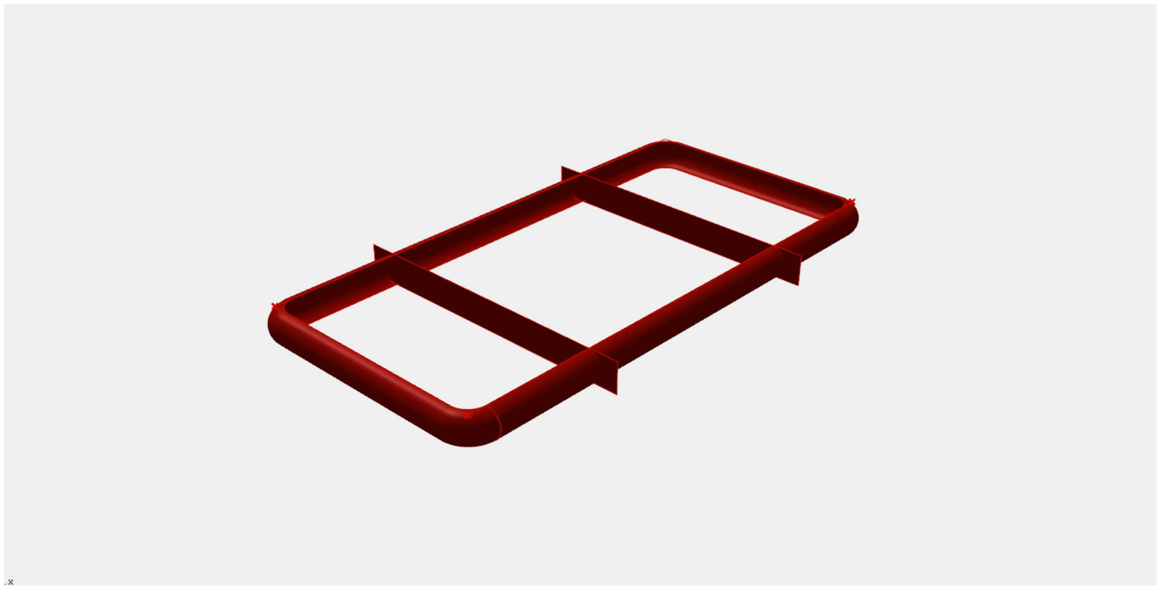


Figure 76: Swept surface with two intersecting surfaces.

These surfaces will be used as inputs for a Split BREP component, which takes in a BREP to be split, and a surface which will define the location of the split. The output of the Split BREP component includes each piece of split geometry in a list. For this case, the BREP to be split is the swept surface, and the splitting geometry will be each of the two surfaces. The Split BREP component can only receive one "Cutter" geometry at a time, so the Split BREP component is used twice, once on the initial cut, and then again with the appropriate remaining surface from the first Split BREP command. In order to select an item out of a list, a List Item component is used. It uses a list and index number as inputs in order to output the item in the list at the desired index. The List Item component is useful for selecting a single item out of a list of items and will be used extensively throughout this project. In this case, it is used to select the desired surfaces which are output from the Split BREP components.

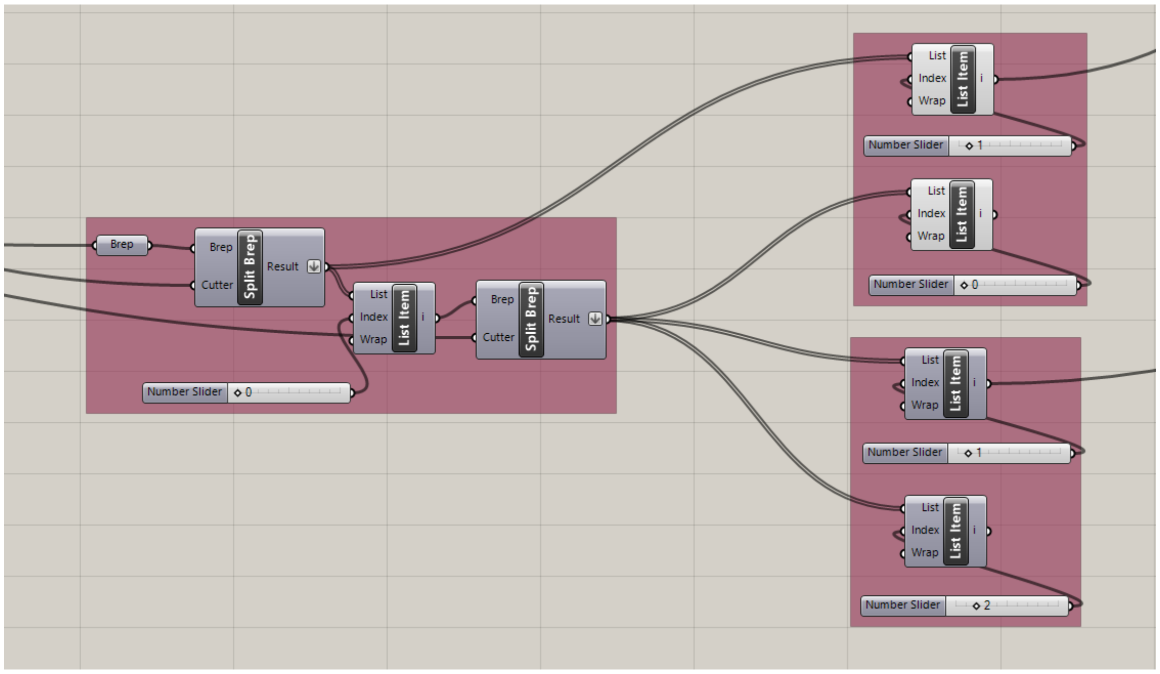


Figure 77: Split BREP components with List Item component selectors

The result of this series of components is two end surfaces to the phone case split at defined locations, shown in Figure 78.

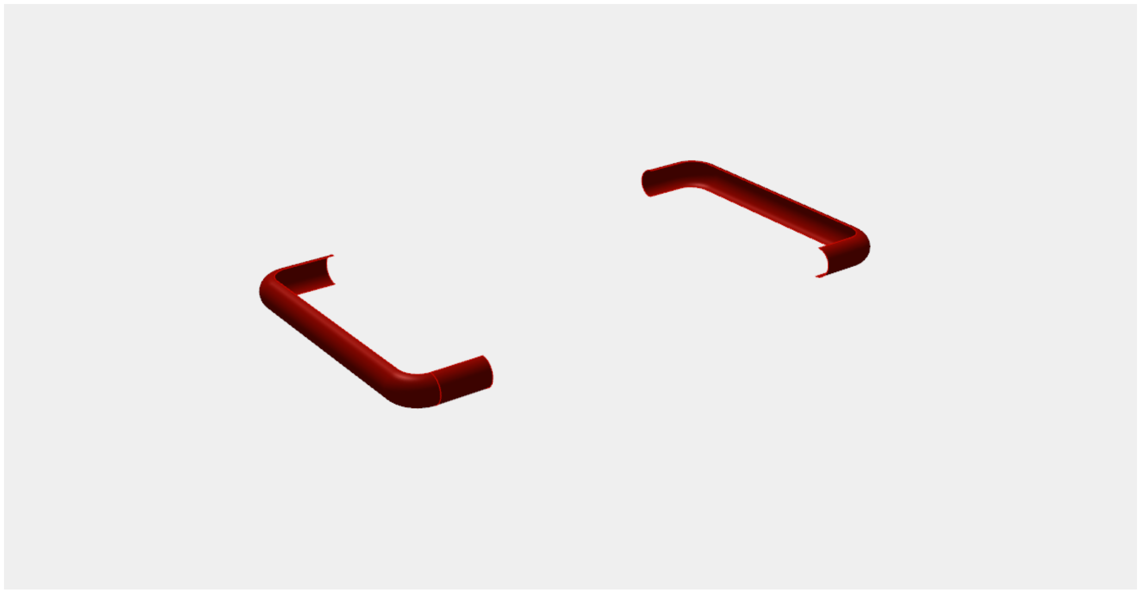


Figure 78: Two end surfaces of the phone case

Now that these surfaces are made, the edges of the surfaces can be used to define the curved surface in the middle. To do this, points at the ends of the edges will need to be selected and then a line will be constructed between these points. To select the desired edges of these surfaces, the Deconstruct BREP component is used, which takes in a BREP and outputs lists of all the contained Faces, Edges, and Vertices of the geometry. One could be curious as to why one would select the Edges of the surface to get specific corner points instead of just using the defined Vertices. A difficulty of working with this particular algorithmic system is that at different sizes, the direction of certain parameters of surfaces, such as u and v directions or normal directions, could reverse. It was found that at different Split BREP locations, the selected points from the Vertices lists would flip sides, creating an inconsistent algorithmic code. To work around this, the edges are selected, then the Curve Closest Point component is used, which takes in a curve and a defined point, and outputs the point which is located on the curve closest to the defined point. This work-around assures that no matter the size of the surface or surface directions, the same point on the edges are selected each time. For this case, the defined point for the Curve Closest Point component is the origin and the Curve inputs are the desired edges of the surface. The two points created by the Curve Closest Point components are used as start and end points on the Line component, and then this line and selected surface edges are combined with the Join Curves Command. Another surface edge along with this combined curve is used to sweep the final surface.

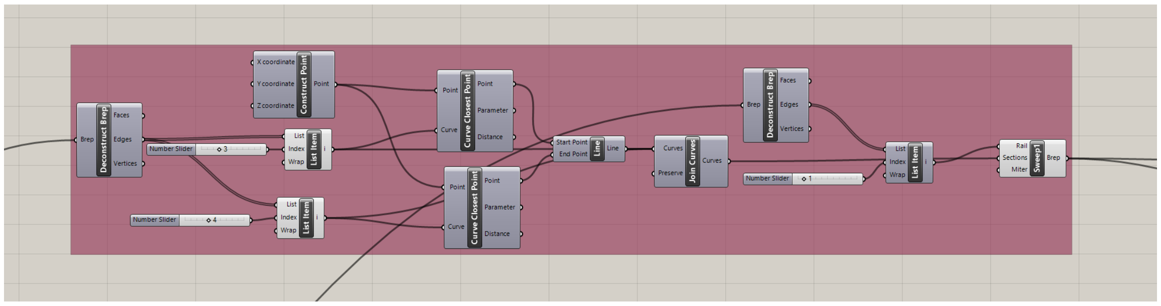


Figure 79: Deconstruct BREP, Curve Closest Point, Join Curves & Sweep Components

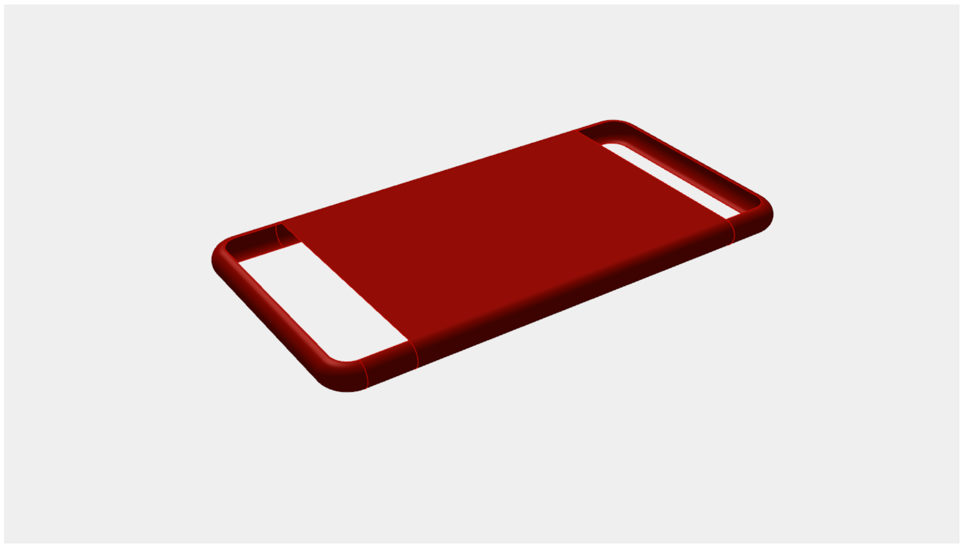


Figure 80: New middle swept surface with end surfaces on the sides

This particular method of creating a surface assures that the surface wraps around the sides of the phone case and matches the curvature of the previously defined sweep surfaces. The next step of this process is to attempt to create isolines along this surface. What is quickly evident, though, is that this particular curved surface has some issues with the distribution of its isocurves, perhaps due to it being created from several joined curves. The components shown in Figure 81 creates isocurves along the curved surface by using a Range component to create a set of numbers evenly spaced from 0 to 1. These numbers then define the V coordinates in an Iso Curve component.

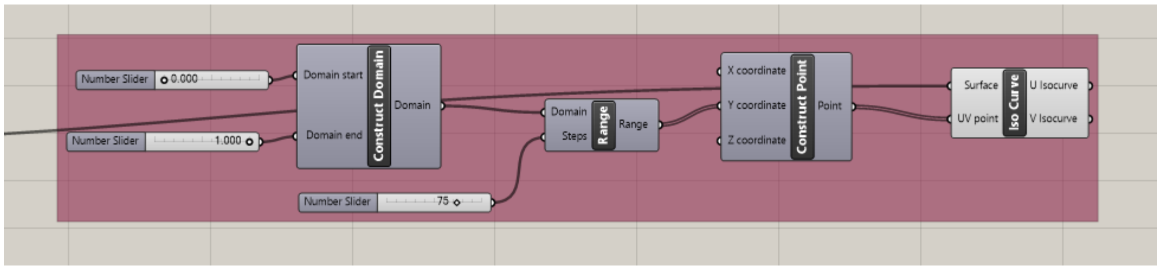


Figure 81: Components creating a set of 75 Iso Curves from 0 to 1 V values.

The result of these components is a series of Iso Curves on the surface, shown in Figure 82. It is apparent that although the isocurves values are evenly spaced, the actual distribution on the surface is not evenly spaced, with far more isocurves located on the curved portions than in the middle sections.

At this point the decision was made to switch methods and use a flat surface instead of a curved surface. This method will not allow for any ridges to be made on the sides of the phone case but may save time and effort in trying to work around this particular problem.

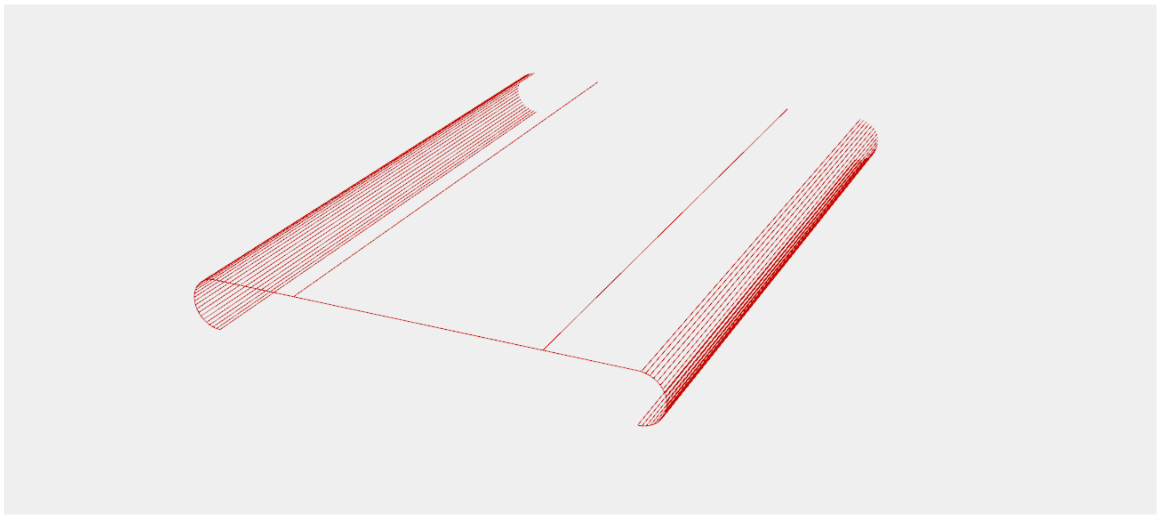


Figure 82: Uneven distribution of Iso Curves on surface

After backtracking a bit to the BREP splitting portion of the code, a similar method is used by inputting the Swept surfaces as the BREP and planar surfaces as splitting tools. This time, additional cuts are made to further subdivide the surfaces. More List Item components are added so that all the pieces of the divided surfaces are contained and easily found later on if needed. This new organization is shown in Figure 83.

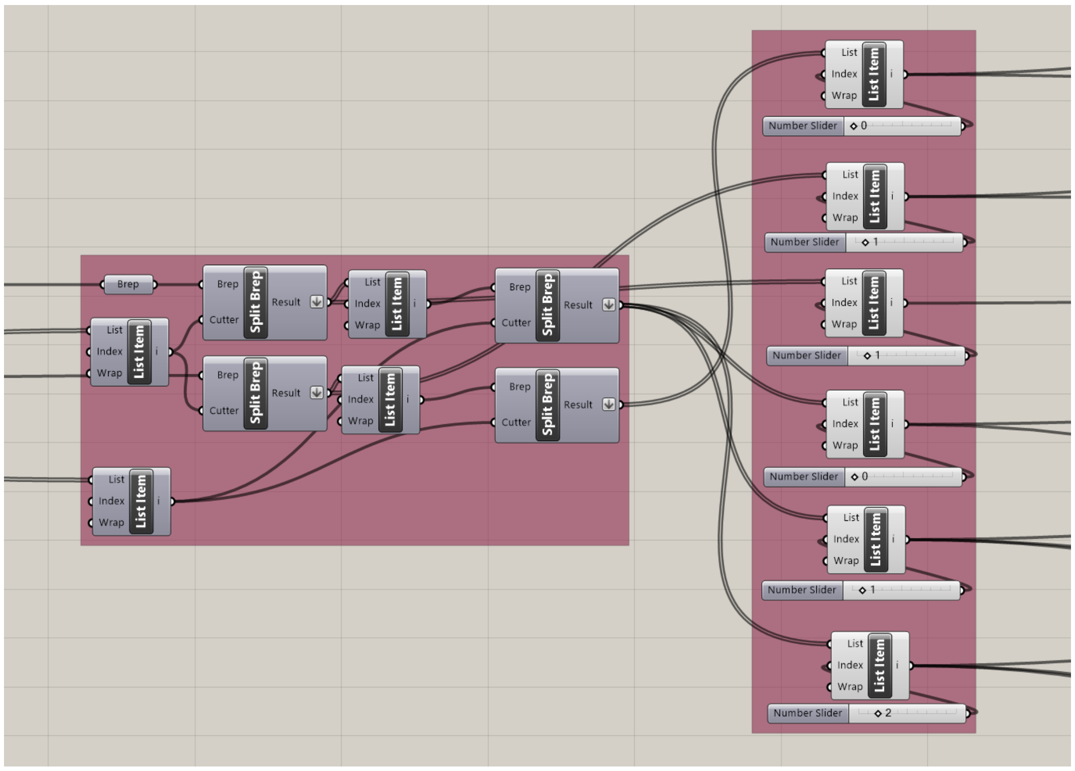


Figure 83: New Split BREP and List Item structure for a flat surface method.

The subdivided and organized surfaces, shown in Figure 84, have only the flat middle work surface left to define, but the edges of the other surfaces will help.

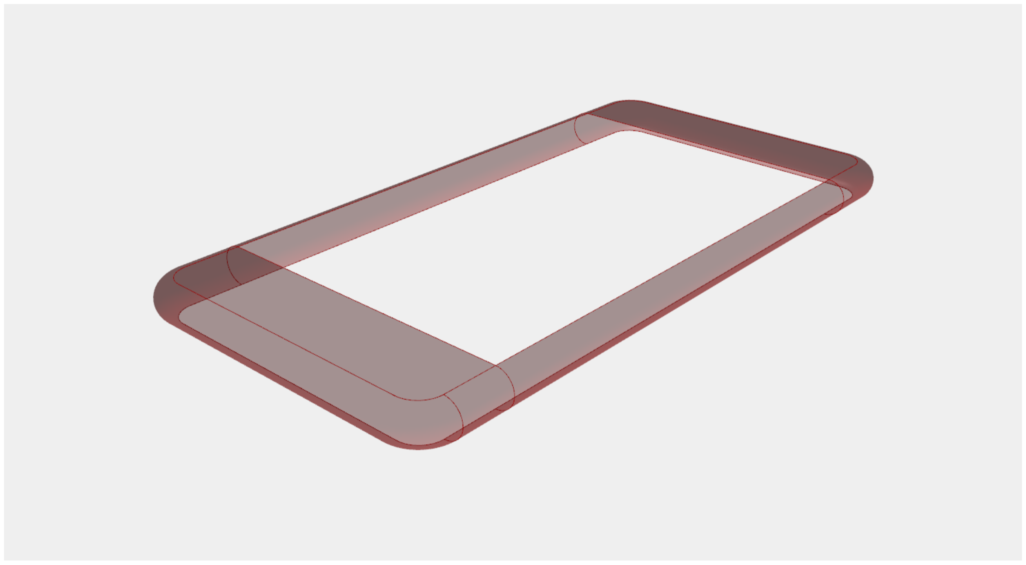


Figure 84: Split BREP surfaces with missing center surface

To get the center surface, the edges of the other surfaces can be found by deconstructing the individual surfaces surrounding the center. These edges are then used as inputs for an Edge Surface component, which creates a planar surface from a series of input edge parameters. Additionally, the other surface pieces are collected into a single Geometry component, shown in Figure 85. For the time being, these surfaces aren't going to be used, and the rest of the algorithm will focus on the newly created Edge Surface. It's also worth noting how, at this point, the algorithm has almost contracted into a single component. Organizing and narrowing down the focus of the algorithm periodically, when possible, is helpful for understanding the next steps in the process, and for creating branching points for diverging designs. Now that everything is organized and the surfaces have been created, this section could be revisited later with a different design intention in mind. For now, the algorithm will attempt to continue following the original design intent.

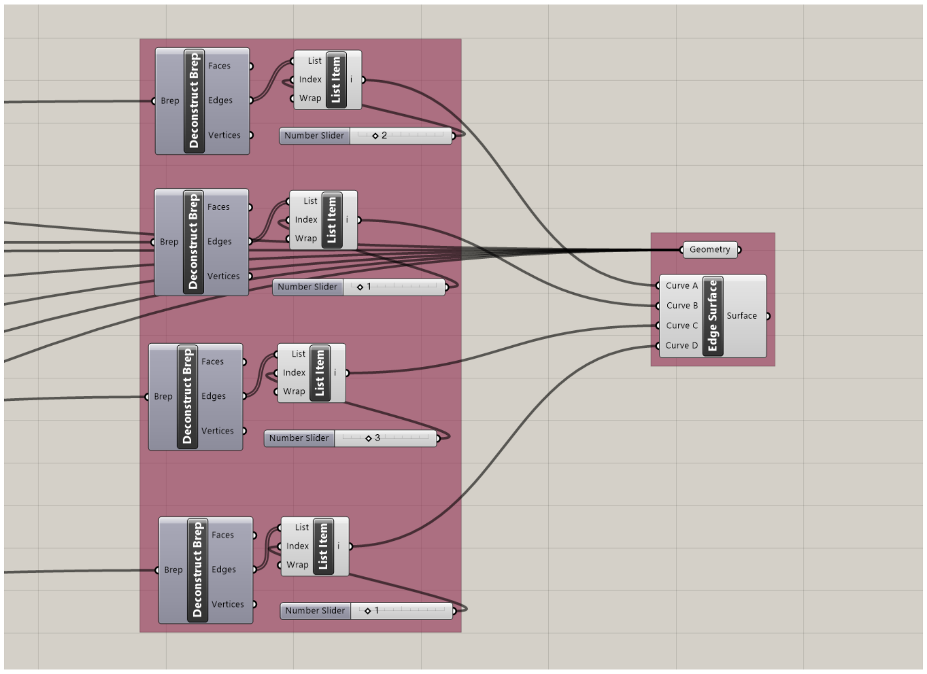


Figure 85: Creating the Edge Surface and organizing other surfaces.

The result of the most recently created components is shown in Figure 86, with the collected surface geometries shown in red, and the new edge surface workpiece shown in green.

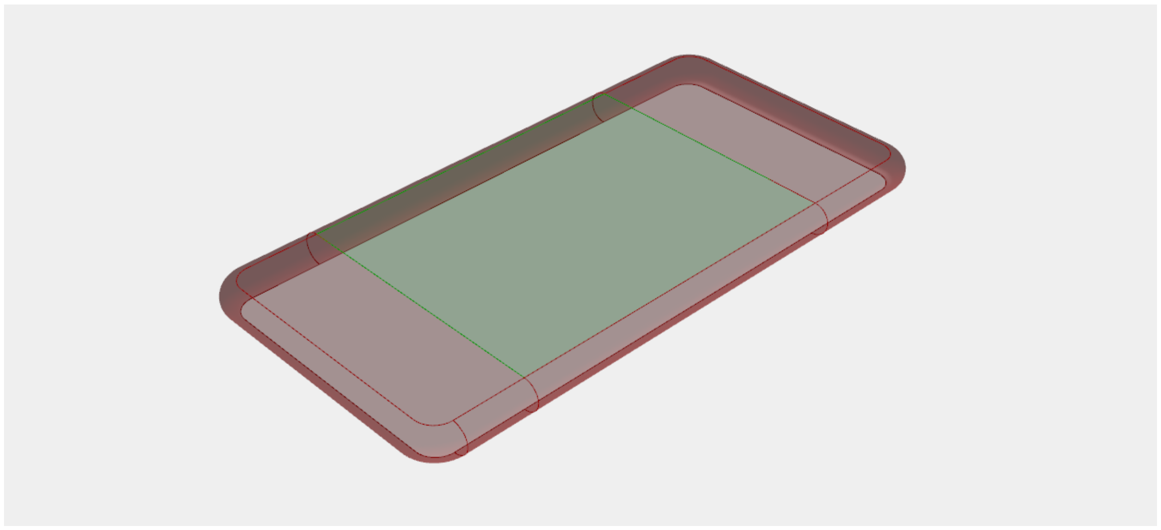


Figure 86: Collected surface pieces and planar surface workpiece

Now, everything but the planar surface workpiece will be hidden so it, alone, can be focused on. If one recalls the abstraction plan, the next set of steps will create lines along the sides and middle of the surface with variable sizes. To begin, the edges of the surface are found by deconstructing the BREP, and a middle line is defined by an isocurve at where U and V equal .5. This process is shown in Figure 87. What is also shown are additional Reparameterize Curve components which take the global coordinates of the line input and output a line whose points can be found from a parameter from 0 to 1.

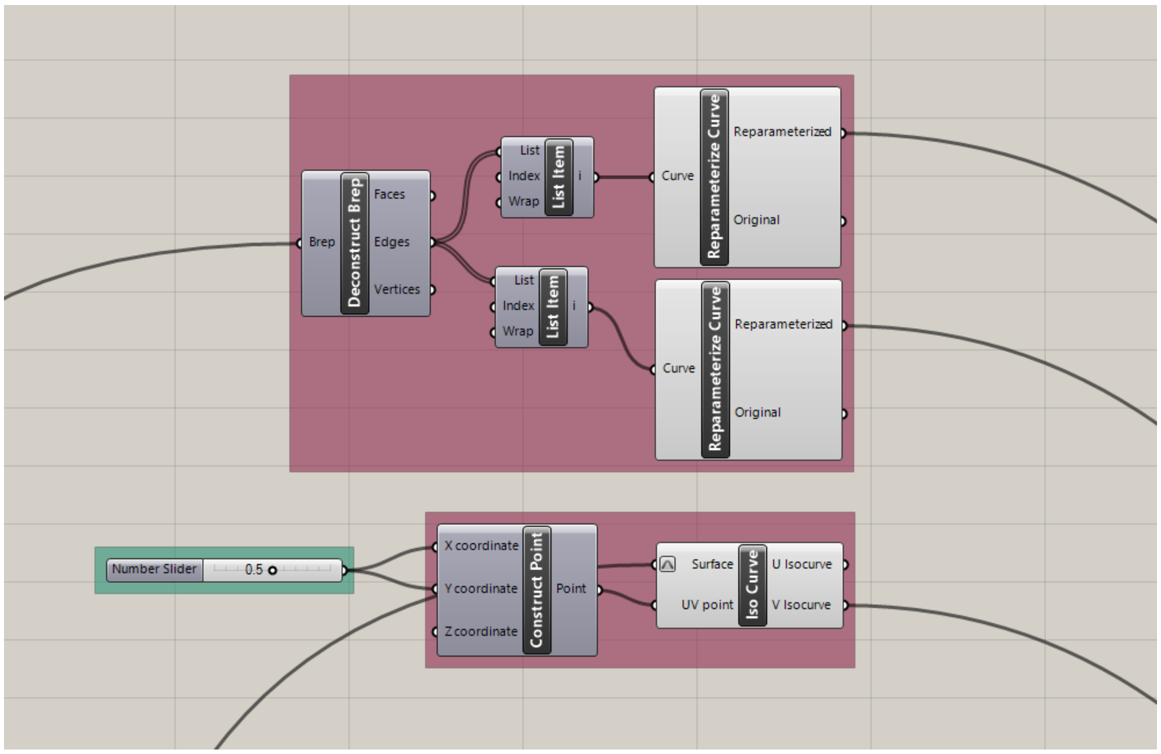


Figure 87: Creating lines on sides and middle of surface.

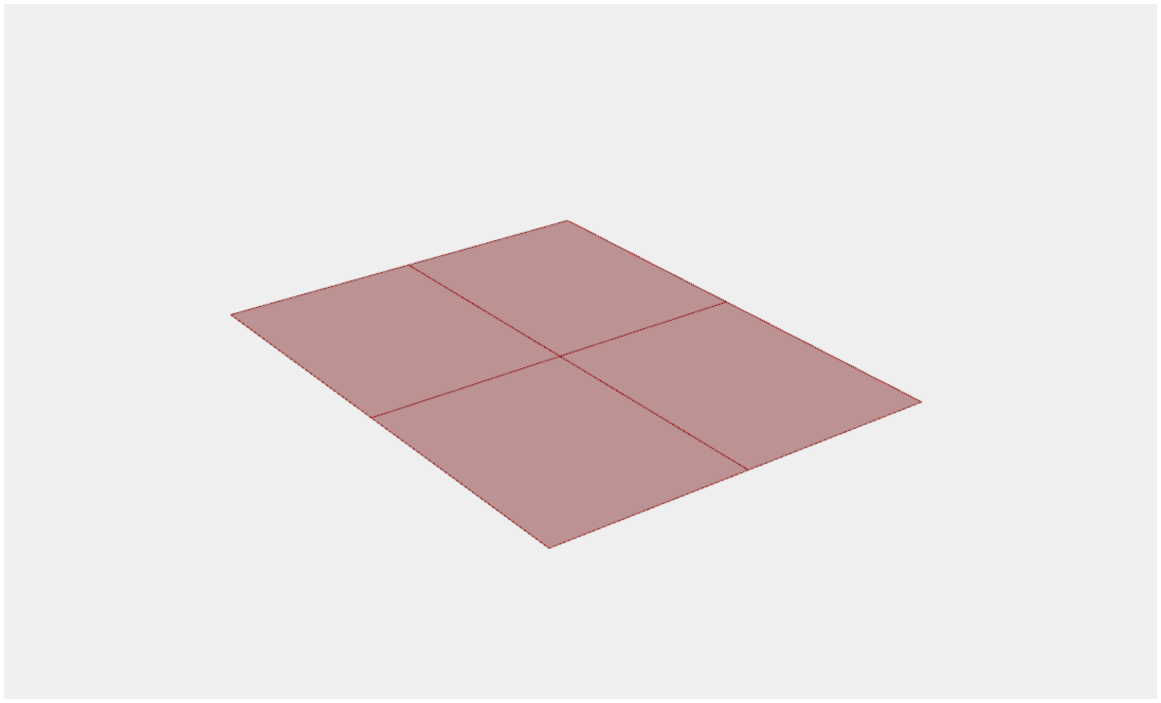


Figure 88: Surface with lines on sides and middle of surface

Now that the lines have been established, a system to adjust the size of the lines must be created. Using a VPL, and coding in general, offers many different avenues to accomplish the same action. At first glance it may seem that the easy solution would be to use a Scale component along an axis to change the size of the lines. This could work, but the design intent is to be able to set a distance from each edge. So instead, a Shatter component is used. Shatter takes in a line and a parameter distance along its length and splits the line at that point. By splitting each line using two points, three shattered lines are created. By selecting the middle line, a line defined by a distance from each edge is created. This line is then run through a Divide Curve component, which divides a curve into a specified number of points. In order to transfer the points from the lines to points on the surface a Surface Closest Point component is used. This system of creating lines and defining points on the lines is shown in Figure 89.

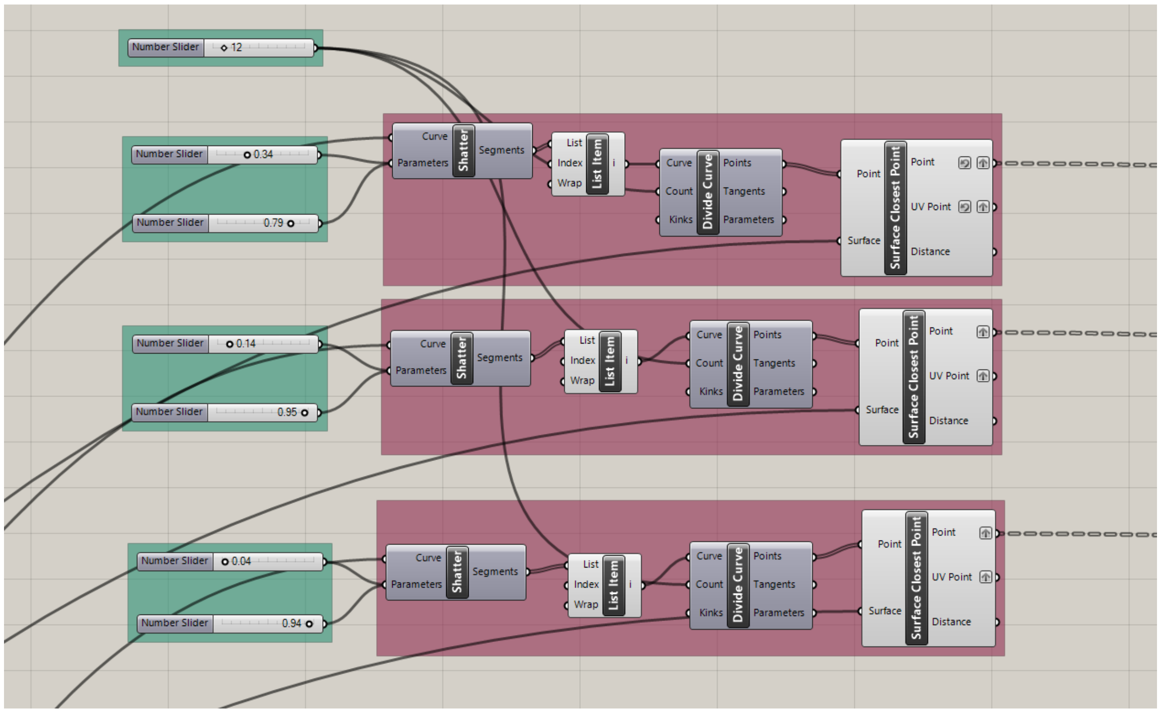


Figure 89: Defining lines and dividing them into points.

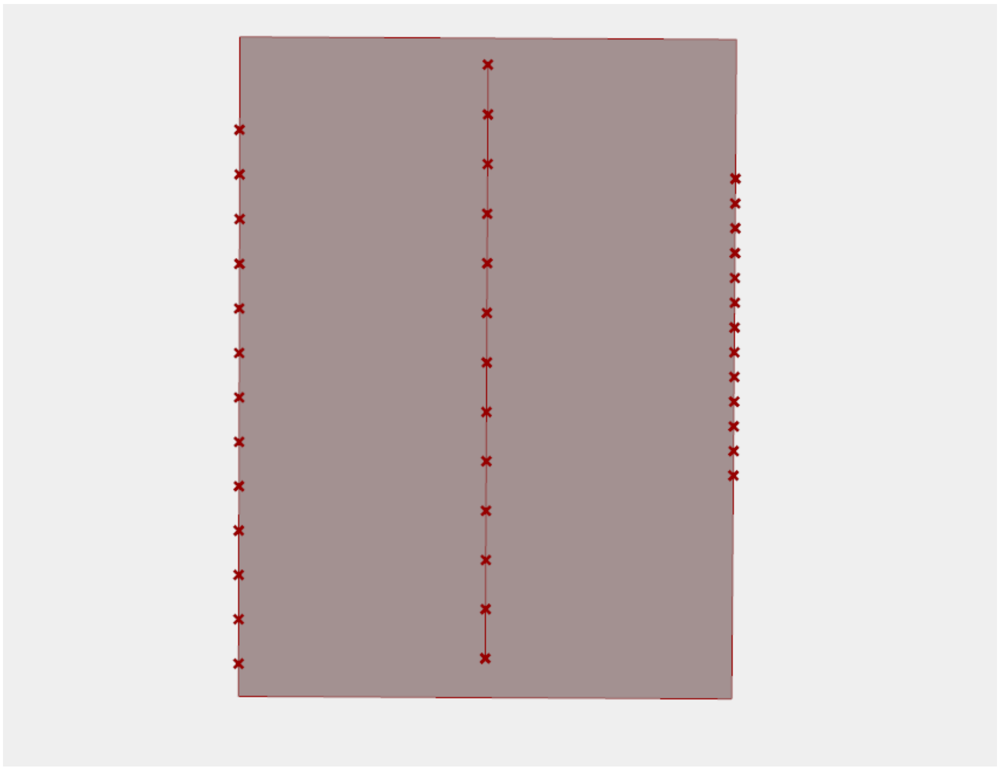


Figure 90: Truncated lines along the edges of the work surface with point divisions.

In the algorithmic abstraction, splines were added at this time to connect the points together. Instead, this system will move the center points vertically before attempting to create a spline between them. Adding the splines now and after the points are moved would be a bit redundant so they will just be moved for now.

To move the points to form the ridge pattern, every other point must be moved a specified distance in the positive Z direction. First, a patterned list must be created. By using a Merge component, to combine the desired distance and 0, a two-item list is created. This list is repeated with a Repeat Data component which runs for the length of the number of points. This creates a list of alternating values between the desired moving distance and 0, corresponding to each point. This distance list is used as an input for a Z vector which is applied to the Motion input of a Move component. With an input of the point list, each point as a translation applied to it based on the repeated pattern. The result is that every other point is moved in the vertical direction.

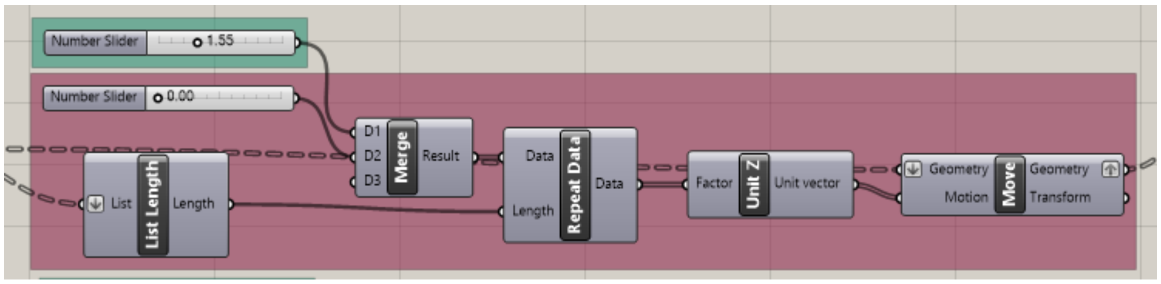


Figure 91: Repeat Data and Move Geometry to create alternating movements

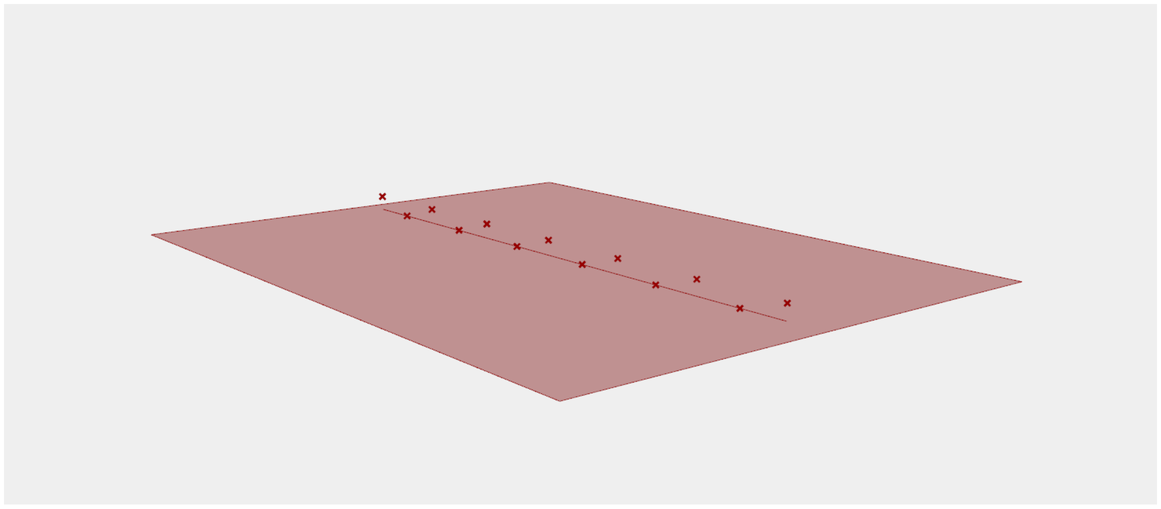


Figure 92: Points moved vertically in an alternating pattern.

At this point it was decided that the system could use an optional expansion beyond what the original abstraction showed. The set of points in the middle of the surface would define the center point for the splines. Since splines can take in multiple points, why not add more lines and points in between the edges and the center? This could create more complex shapes and could be easily turned off if the user preferred the original option. Where the middle line was created by an isocurve at .5 along the surface, two more isocurves will be added at .2 and .8. Its lines will also be truncated and split into a series of points. Now there are five total lines; three in the middle of the surface and two lines on the edges. The code from Figure 91 will be repeated three times for each set of the middle points. The points along the sides do not undergo any transformation, so the code is not applied to them. These transformation sequences are organized in Figure 93, where Geometry components containing each set of points are lined up in order on the left, fed into needed transformation systems, and then output to a geometry node on the right. To help keep track of the points, the sets of points are organized sequentially, with the first edge corresponding to the top row of geometry, the first set of points at .2

isocurve as the second row, the middle line at .5 isocurve in the middle of the stack, then the .8 isocurve line next, finishing with the opposite edge curve on the bottom. This system may look complicated at first glance, but reading each row from left to right shows that it simply transforms the middle three lines in ways which have already been explained.

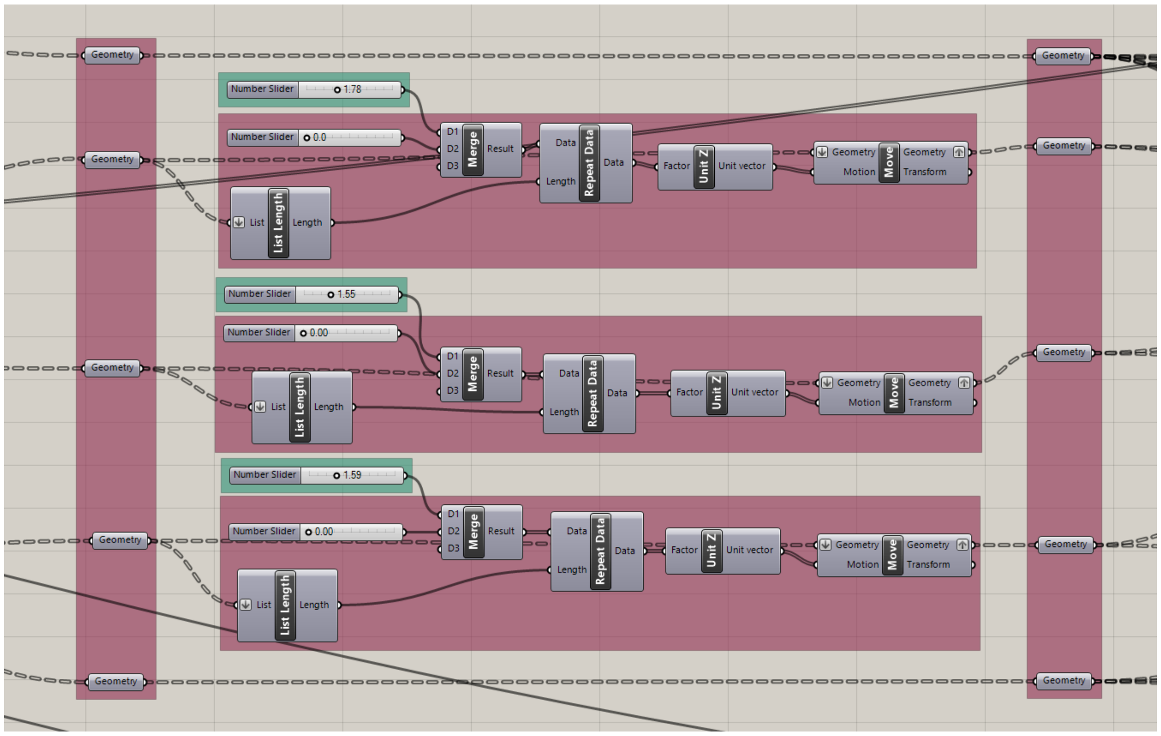


Figure 93: Transformation systems for each line on the surface.

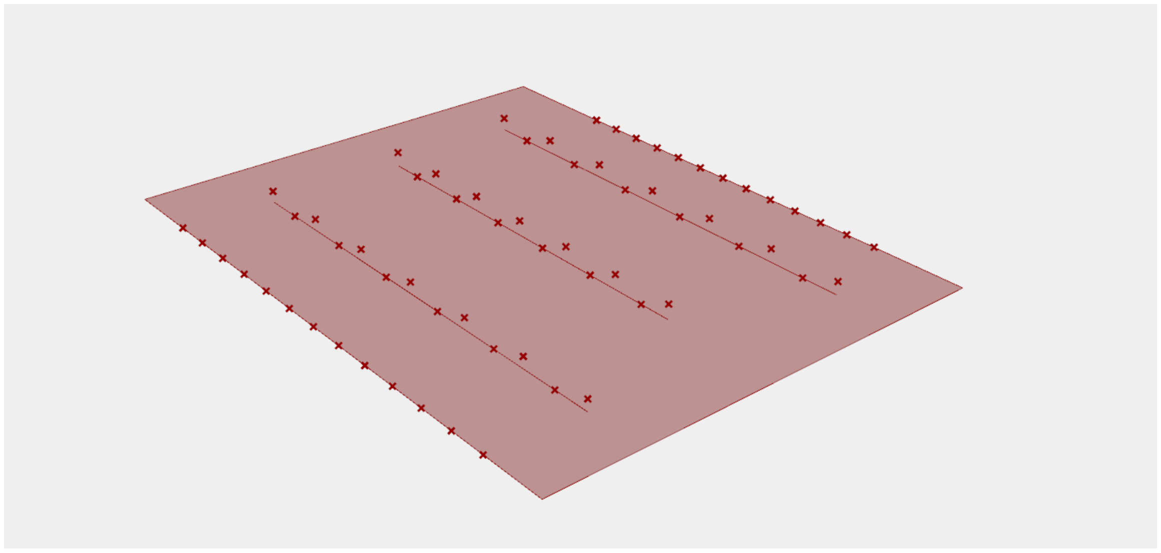


Figure 94: Complete set of transformed points with newly added lines.

Now that the points have been fully defined, the splines can be run through each set of points. Figure 95 shows how the Interpolate (tangency) component is used to accomplish this. For this example, the two additional lines which were added in the previous section have been disabled for simplicity; their nodes and connecting lines have now turned orange and can be ignored. The Interpolate component takes in a set of vertices to run the spline through. In this example, the lists of points from the first edge, the middle line, and the final edge are placed into the vertices input. Since it is taking in three different lists of the same length, it will apply the Interpolate component to the corresponding items in each list. So, the first items in each list have a spline put through each of them, as do the second, third and so on. This Interpolate component allows for tangencies to be defined as well. For the sake of a better design it would be beneficial to have the ridged surface match with the surrounding surfaces as smoothly as possible. The tangents will be defined as the Y direction so that the splines end and start perpendicularly to the sides of the surface. The Match Tree component shown is a tool which allows for the tree structure of a list to be remapped to match a chosen list. This was done because, during the movement and organizing process, the middle list of points' tree structure changed. Matching this changed structure to the unchanged edge points lists is a simple way of making sure the lists will have the same structure before entering into the Interpolate component.

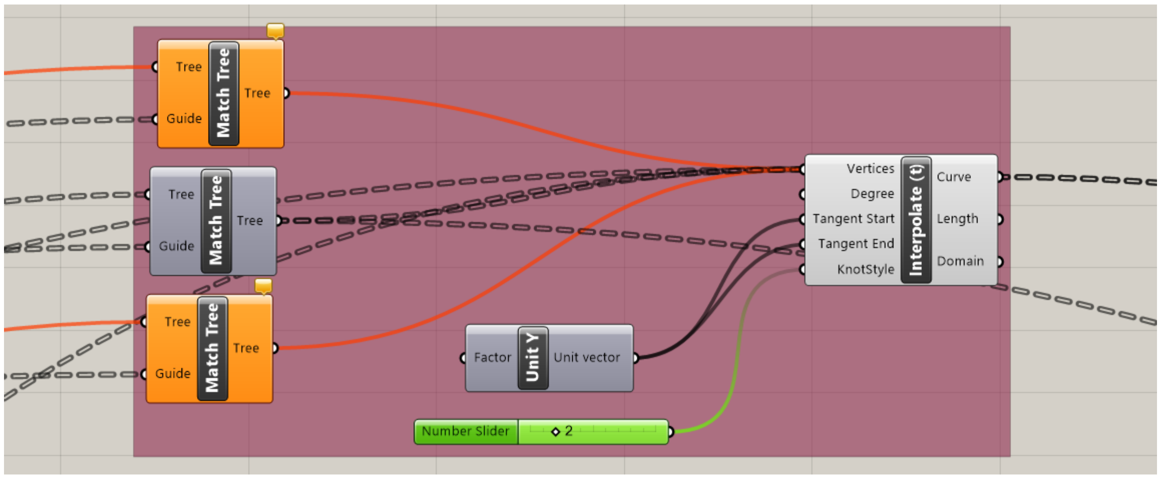


Figure 95: Interpolate component creating splines between edges & middle points

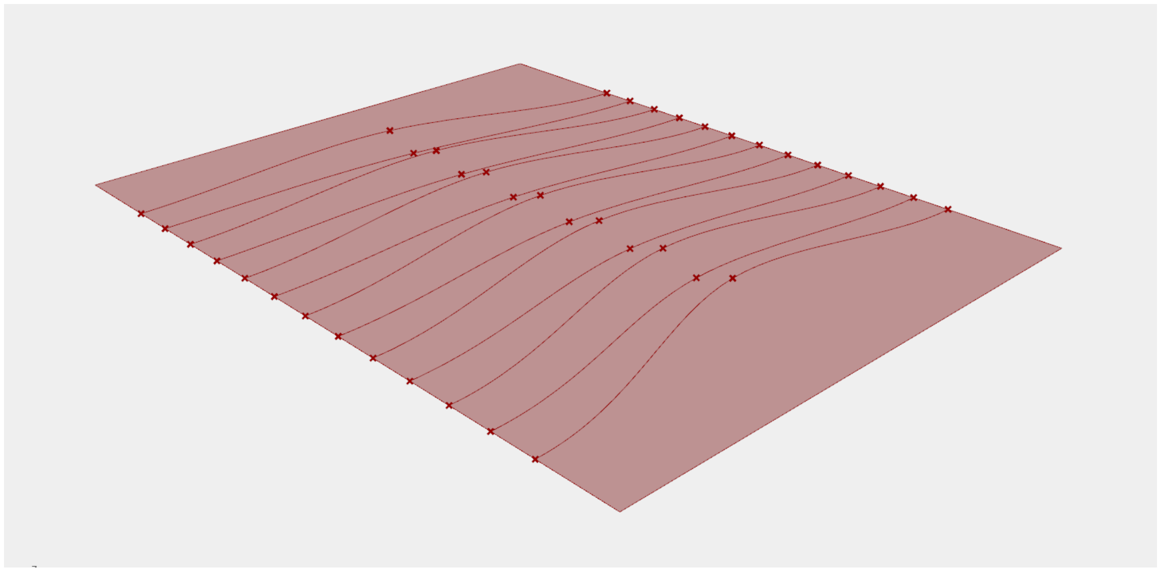


Figure 96: Splines through edge and middle points on surface

To illustrate how the system looks with the additional lines, they have been reactivated for Figure 97. The additional lines add more complexity but create more variability between designs.

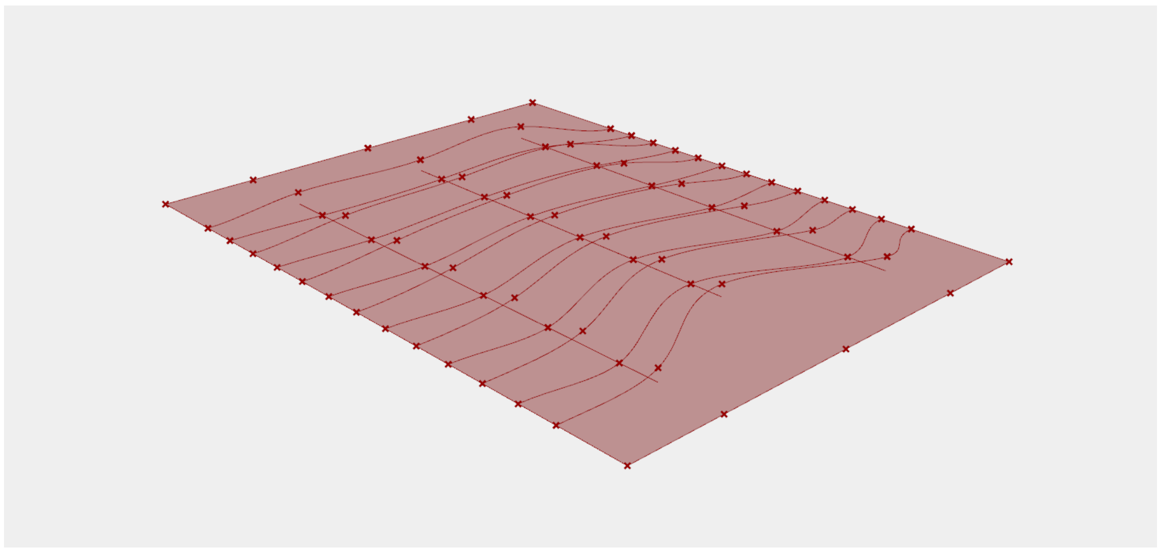


Figure 97: Splines through edge and all middle points on surface

Now that the splines in one direction have been created, the splines in the other direction between each of the points must be created as well. This is accomplished in Figure 98, which is the most complicated looking set of components so far but is actually rather simple. The top and bottom edges must be divided along their length to match the locations of the truncated lines created earlier; this is done using several Point on Curve components shown in the first red box on the left in Figure 98. These points must be inserted in the list of points along each truncated line at the beginnings and ends. The second red box in Figure 98 shows how the Insert Item components are used to do this. Insert Item takes in a list, an item, and an index inputs and outputs a new list with the same values as the original list but with the input item at the location of the input index.

This process is repeated for each set of points. The final red box in Figure 98 on the right shows a series of Interpolate components which take in each set of points and create a spline through each set. These splines need to be tangent to X direction to maintain as smooth of a transition as possible.

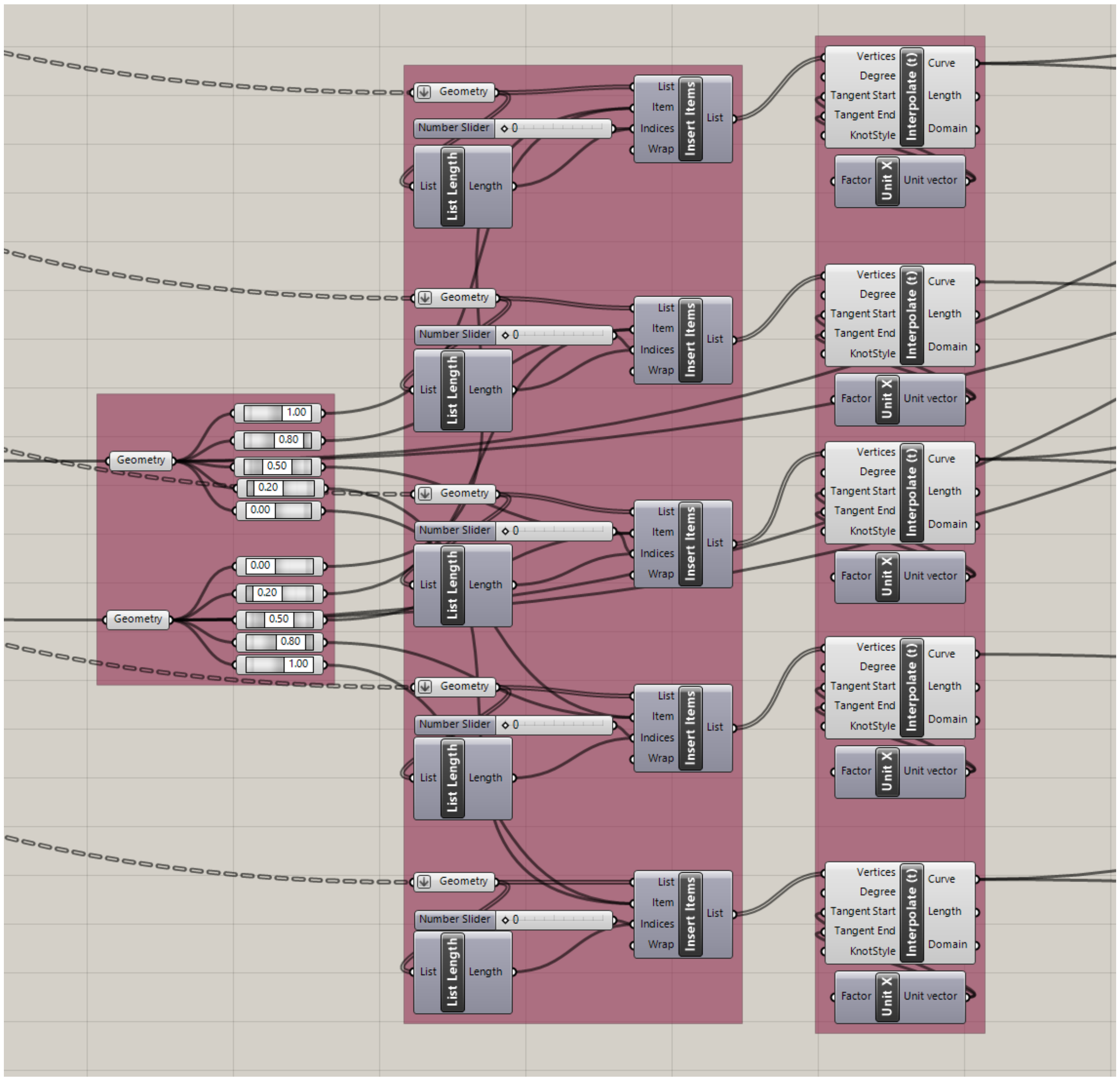


Figure 98: Creating splines along point sets in the other direction.

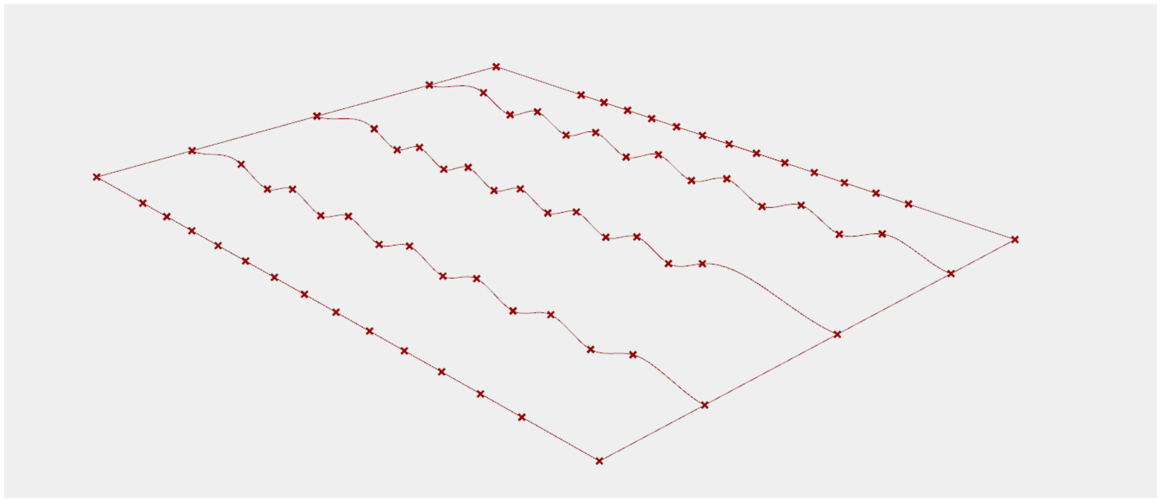


Figure 99: Splines along point sets in the other direction

Splines through the 5-point sets have been created in both the U and V directions. These spline sets are then combined into Curve components for organization purposes. Each set of curves is now used as the U and V inputs for a Network Surface component, which takes in curves from the U direction and the V direction and attempts to create a surface along all of the input curves.

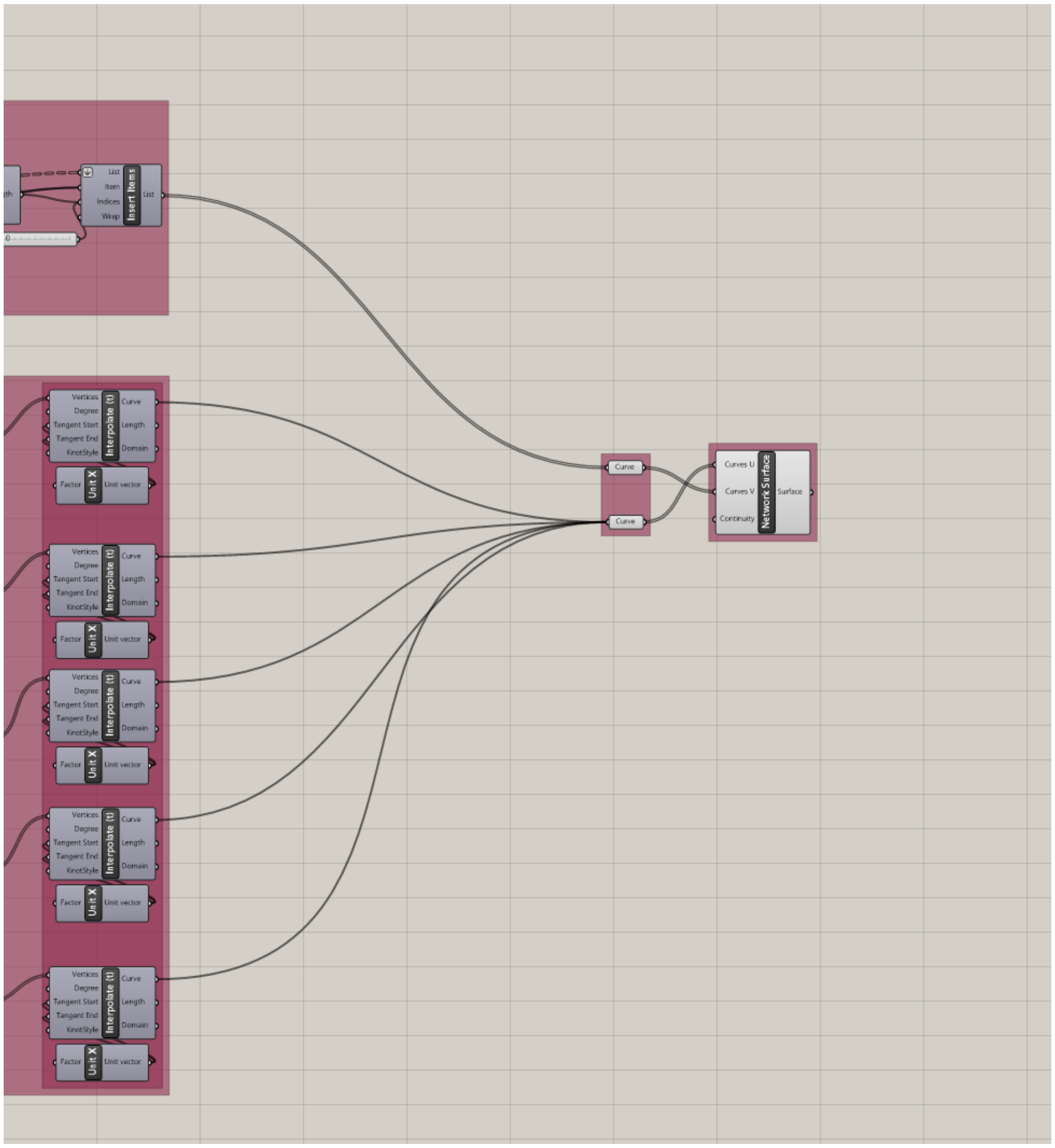


Figure 100: Network Surface component taking in all defined Interpolate splines

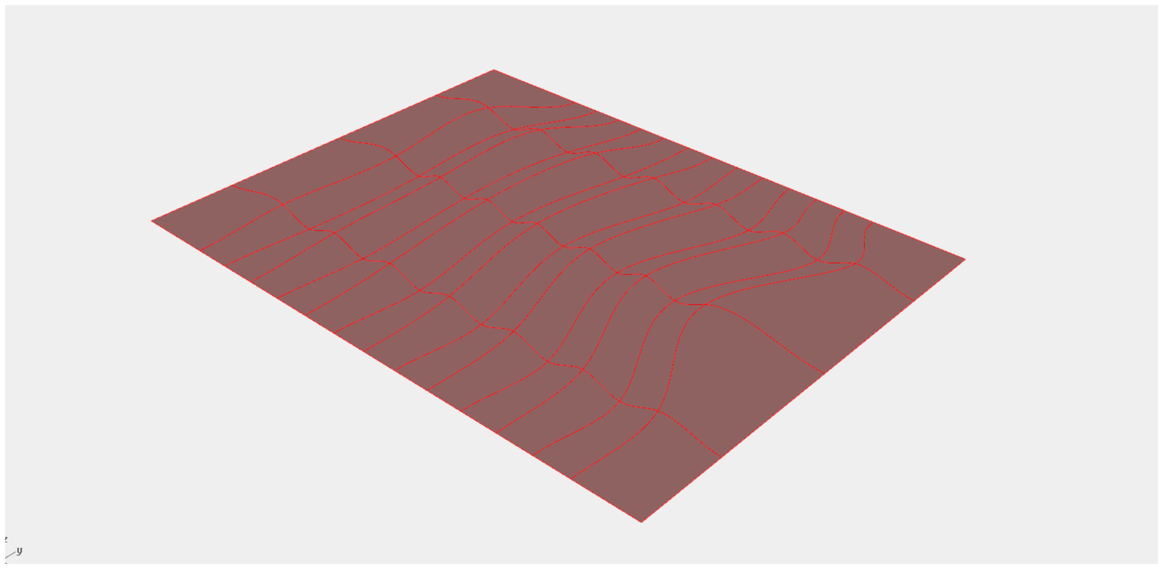


Figure 101: The final ridged network surface with showing interpolate splines.

The surface now has enough variability to fit our abstraction concept, but it can be pushed further. Next, an attractor system will be used to define the height of a ridge point as a function of the distance away from a defined line across the surface; Figure 102 will show this process. The first red box in Figure 102 defines an isocurve along the surface at a certain point using an Iso Curve component. The second box takes in a set of points and measures the distance from the points to this new isocurve with a Distance component. A Remap Numbers component is used to reparameterize the distance of each point to the isocurve as a number from 0 to 1, with points closer to the isocurve being associated with 0 and points further from the isocurve associated with 1. This 0 to 1 range is now multiplied times the height of the Z transformation used for the points, meaning that points closer to the isocurve have their heights multiplied by 0 or a very small number and points further away from the isocurve have their heights multiplied by larger numbers like .9 or 1.

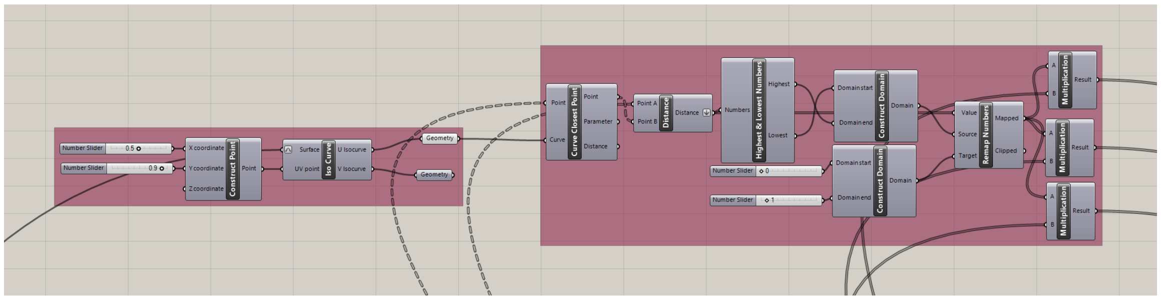


Figure 102: Creating an isocurve and using the Remap Numbers component

If the defined isocurve is set all the way to one edge of the surface, it can be seen how this attractor affects the heights of the points as they get further away from the curve. This is shown in a side profile in Figure 103.

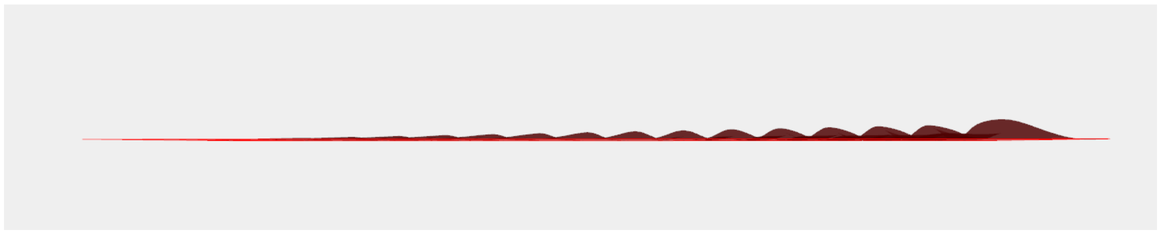


Figure 103: Side profile showing the effect of the attractor system.

Now all the components have been put in place to create the full form of the surfaces which make up the phone case. By showing the ridged surface as well as the other previously created surfaces, the whole form can be viewed in Figure 104.

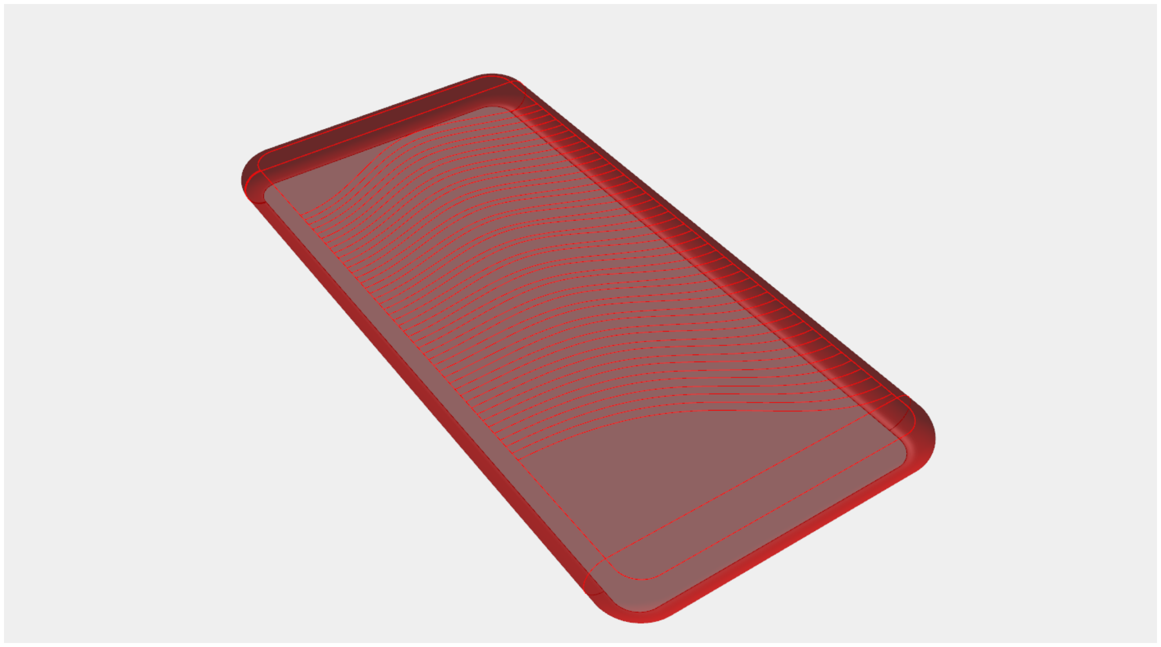


Figure 104: All created surfaces of algorithmic phone case.

Next, the cut-out features for the cameras, power buttons, volume buttons and charging ports must be defined. Each cutout will be created using a slot shaped curve. Each slot is created using two arcs that face each other and have connecting lines between their ends. Using the dimensions of the cell phone size from the very beginning of the algorithm, certain measurements can be taken, with adjustments, to define the location and size of the slot. It was the intention that the algorithm work even for circular features, where the distance between the arcs would be equal to zero. Figure 105 shows the entire slot curve defining process, starting with constants which change depending on the given phone, constructing the arcs, connecting the arcs with lines, and then joining the curves together.

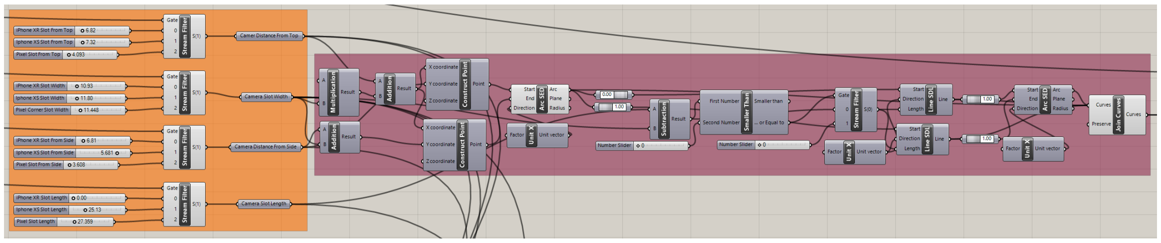


Figure 105: Slot shaped curve defining process

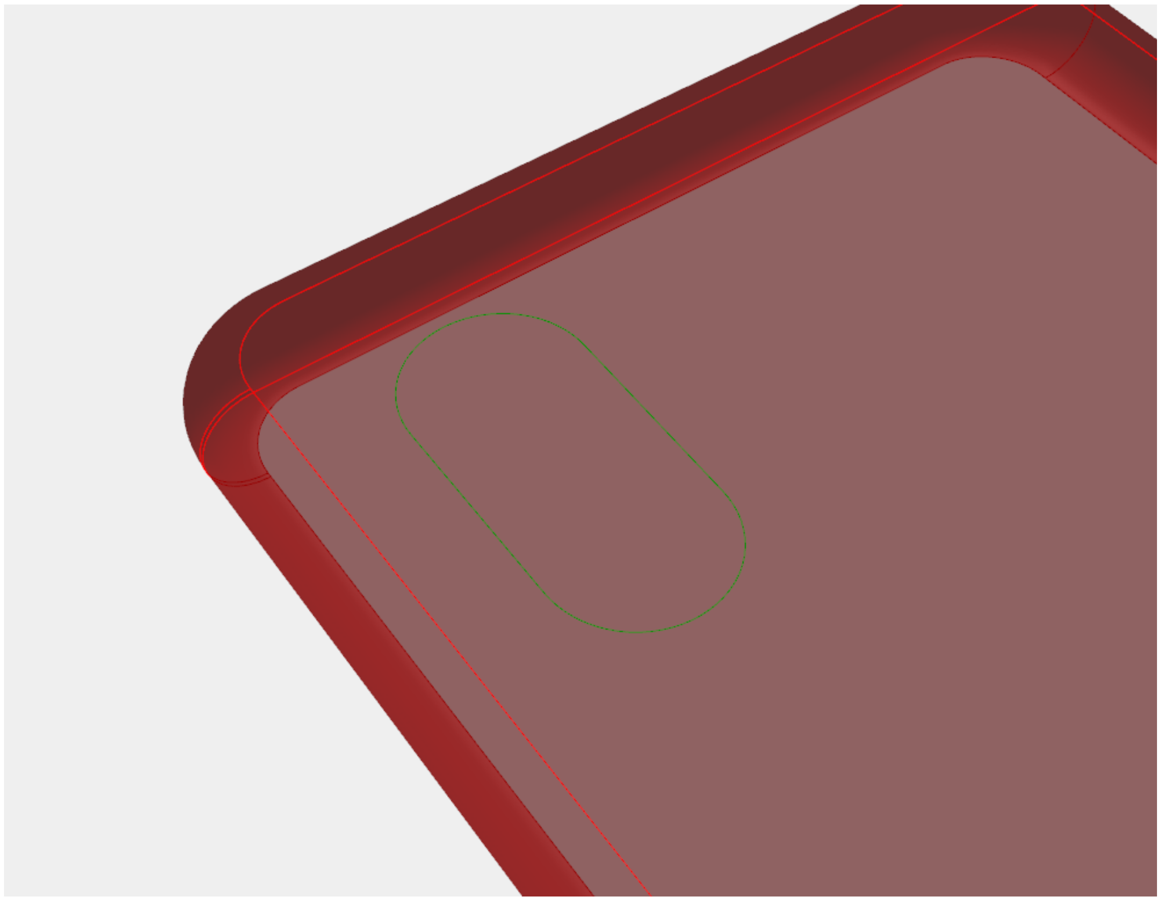


Figure 106: Green camera slot Curve on phone case surface

Once the curve has been defined, it can be extruded from the surface in the +Z and -Z direction, and joined in order to form a surface BREP, shown in Figure 107.

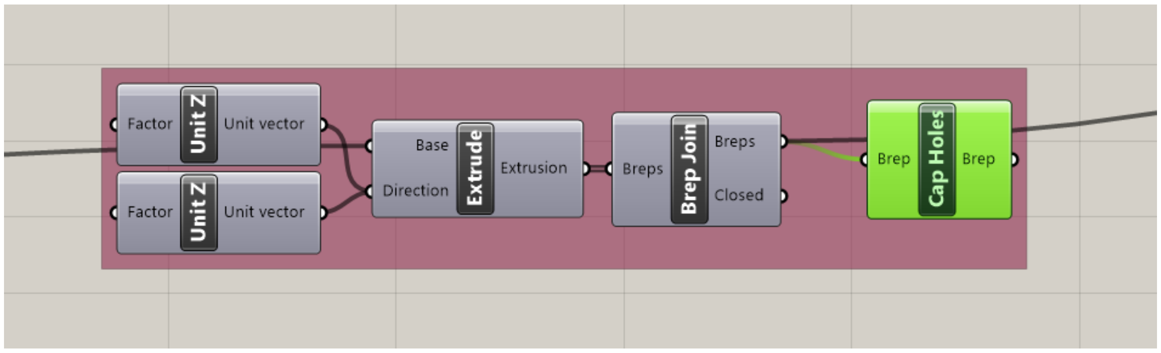


Figure 107: Extruding the slot curve and creating a surface BREP

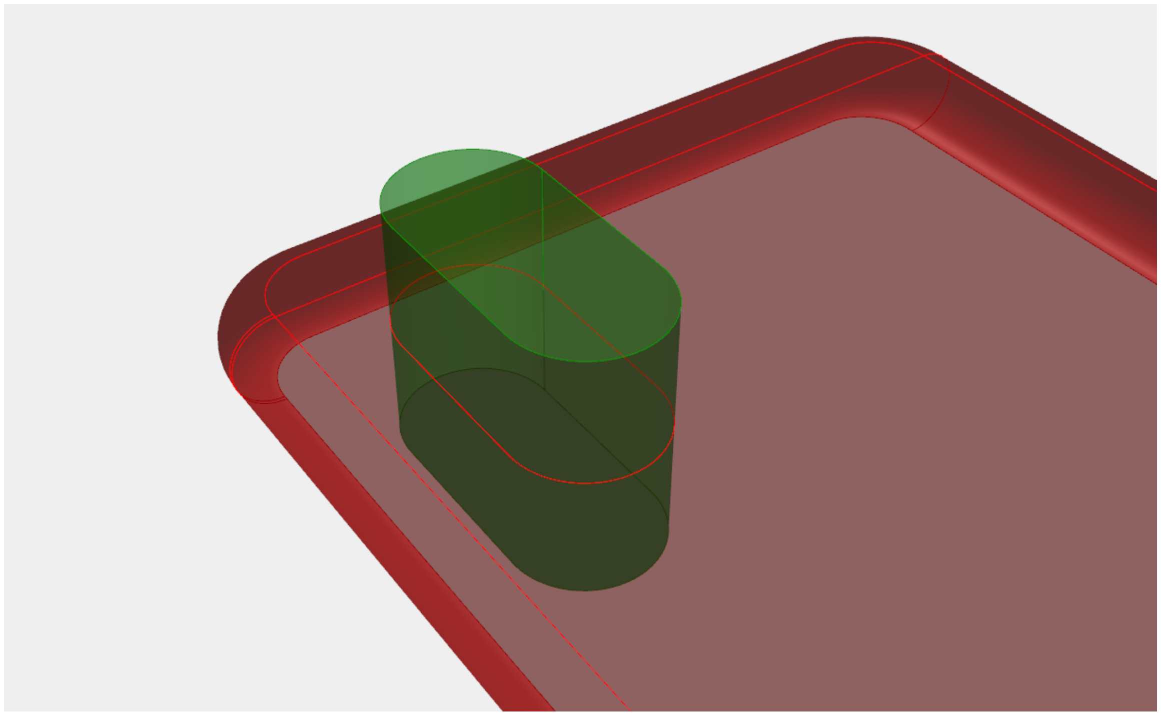


Figure 108: Solid BREP slot extrusion intersecting the phone case.

The slot defining, extruding, and BREP creating process is done for each required cutout. Figure 109 shows the entire system; although it looks complicated, it is the same process shown previously, just repeated for different locations on the phone. Some of the cutouts are only for certain phone types, such as the Google Pixel phone's extra circular cutout in the middle of the phone case that the iPhone models do not have.

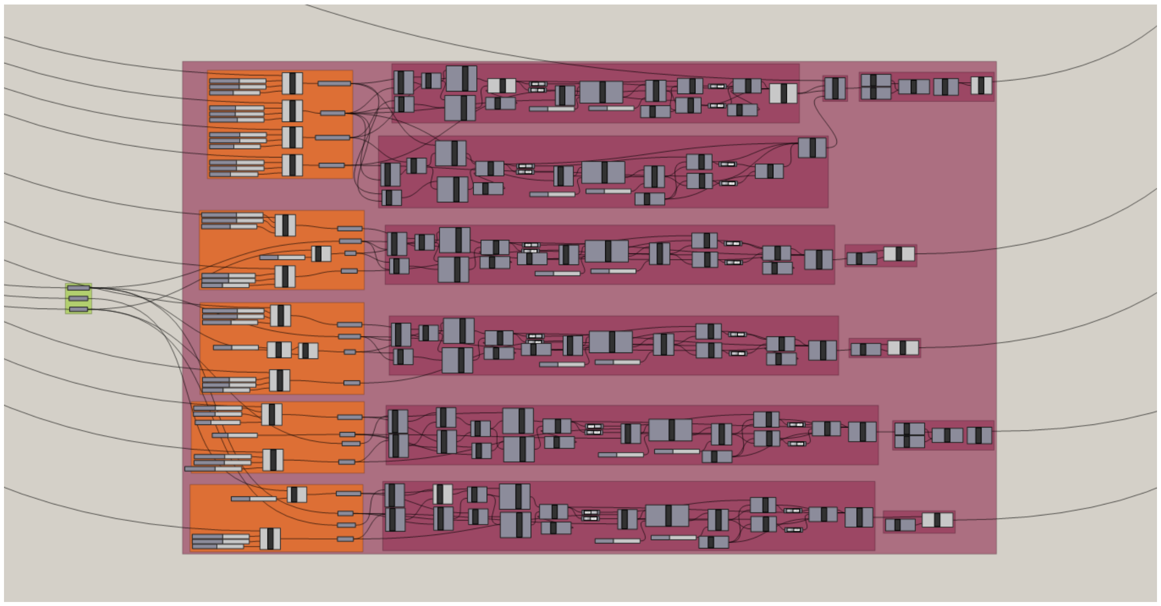


Figure 109: Complete slot extrusion creation system

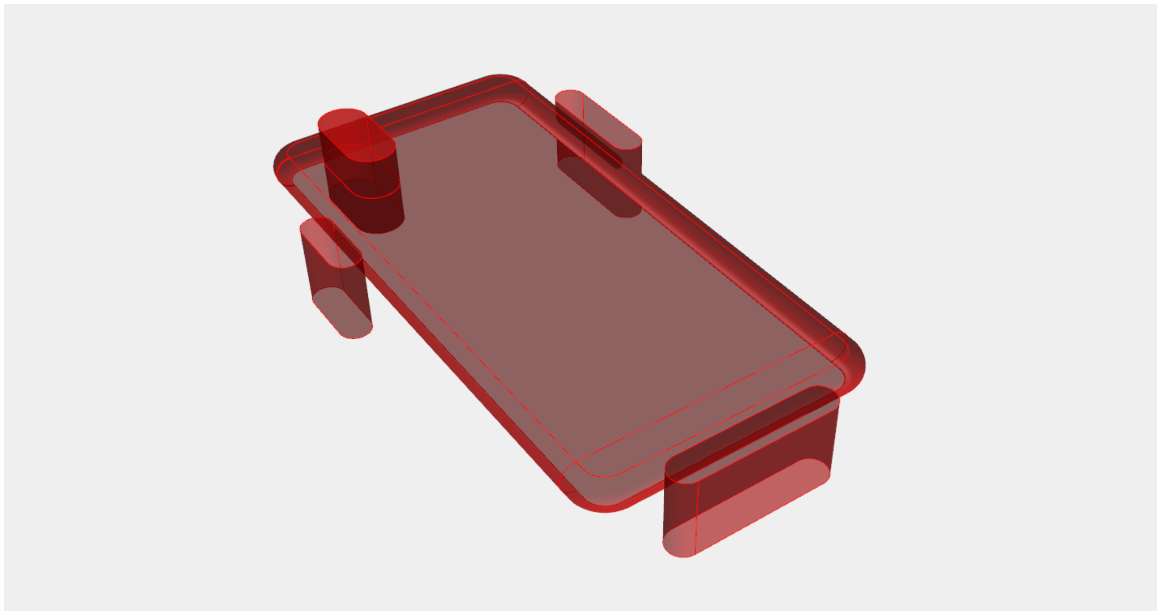


Figure 110: Phone case with all slot extrusions defined.

With all of the extrusions created, they must now be used in conjunction with a series of Split BREP components to remove surface pieces from the phone case. Each usage of the Split BREP component creates a new BREP, which is then used as an input for the next Split BREP component and so on, until all of the extrusions areas have been removed, resulting in the final form shown in Figure 112.

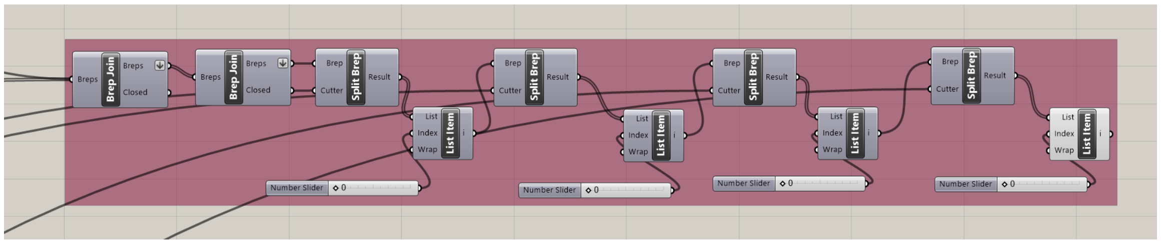


Figure 111: Series of Split BREP components removing slots from the phone case

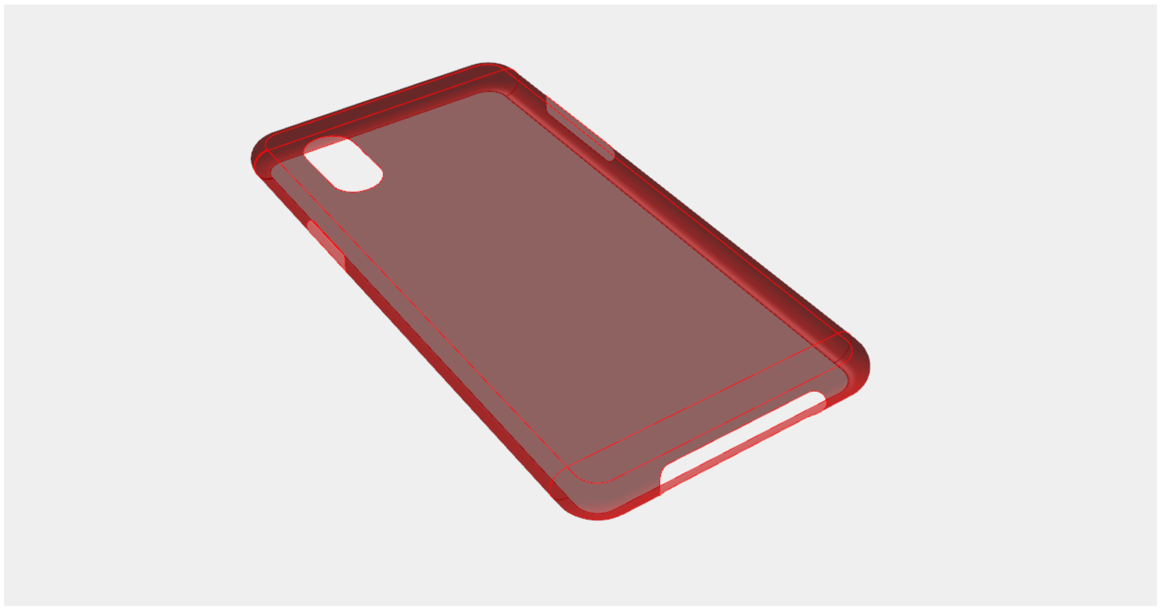


Figure 112: Full algorithmic phone case surface with slots removed

With this final surface, the end of the algorithmic process has been reached for the current design intention. However, as previously mentioned, one of the benefits of the algorithmic design process is the ability to make diverging design paths. This particular design created a work surface on the back of the phone case during the building process that is altered using the ridges and then placed back into the phone case with the rest of the surfaces. With algorithmic design, this work surface can be easily found and used to work towards creating an entirely different design direction. Figure 113 shows the entirety of the algorithmic design in Grasshopper and has a red circle showing where the work surface was created, before any of the ridge creating features were added.

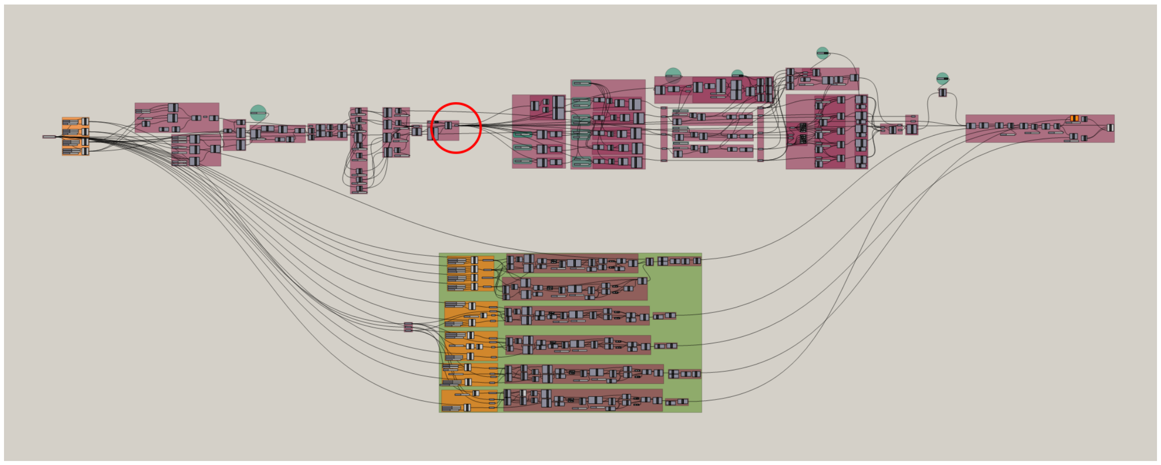


Figure 113: Entire algorithm with work surface highlighted

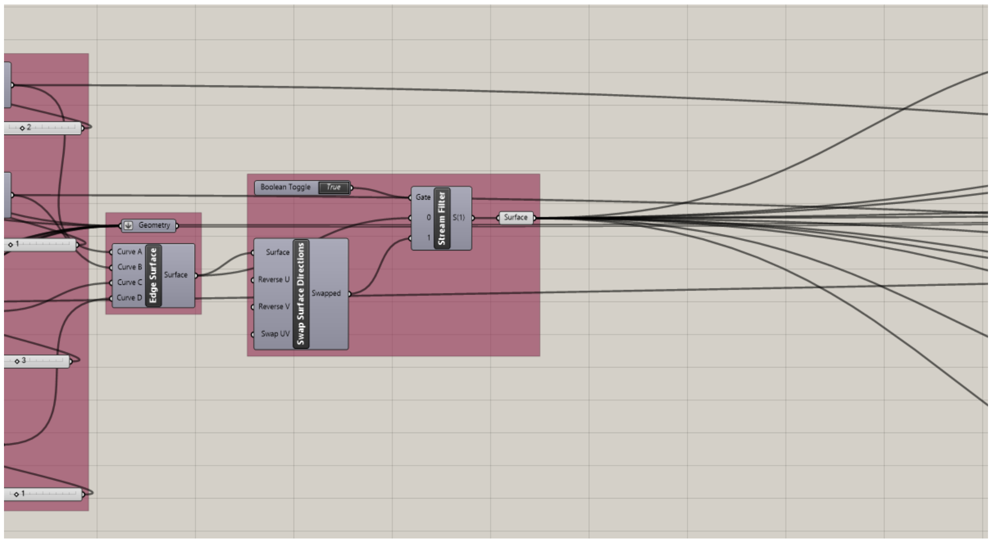


Figure 114: Showing single surface component containing work surface

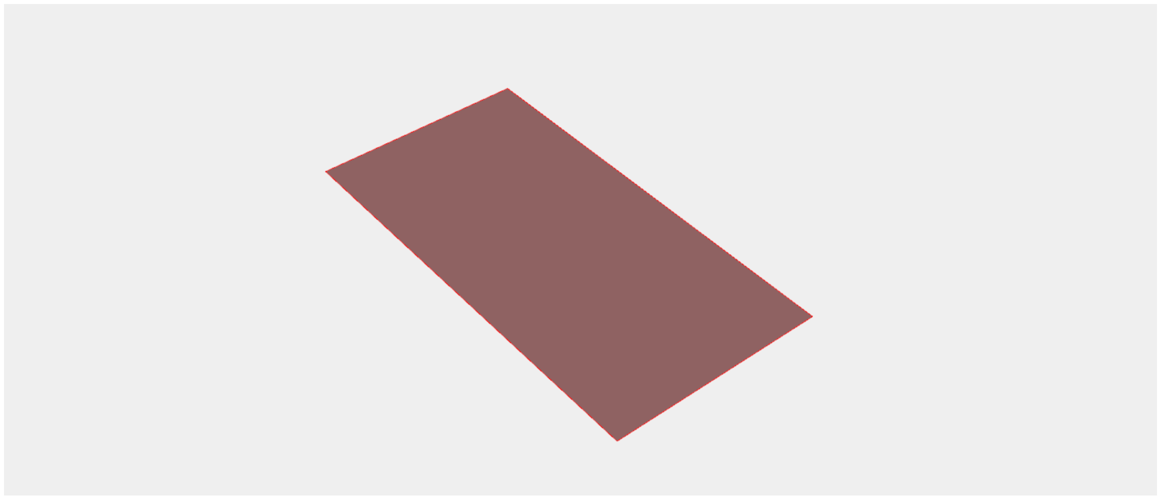


Figure 115: Single work surface before ridges were defined

When this surface was created, it was mentioned that it could be used at a later time for all manner of different designs. An alternate diamond pattern design will be applied to this surface now, and then it will be integrated back into the algorithmic flow. By taking this work surface and using a Diamond Panel component, downloaded as part of the Lunchbox Grasshopper plug in, a diamond grid of a specified number of divisions can be constructed on the surface. Scaling each diamond down around each diamond's center creates a second set of smaller diamonds.

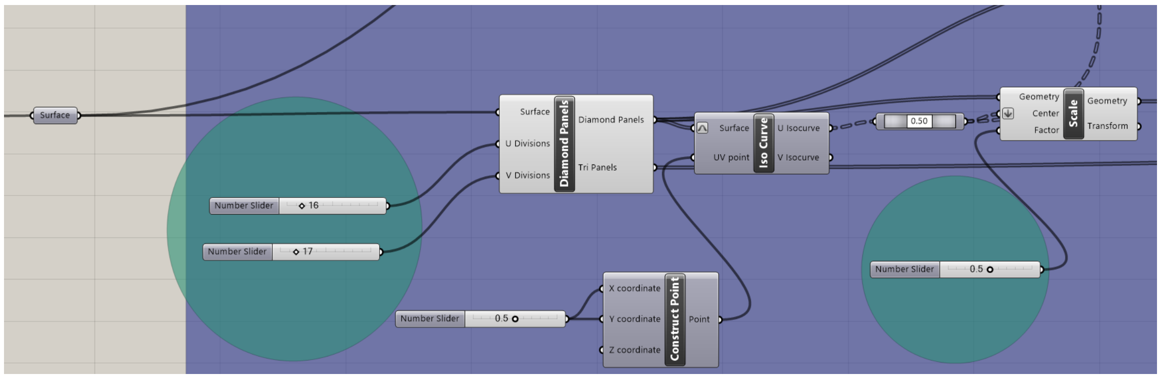


Figure 116: Diamond panel and scaling process.

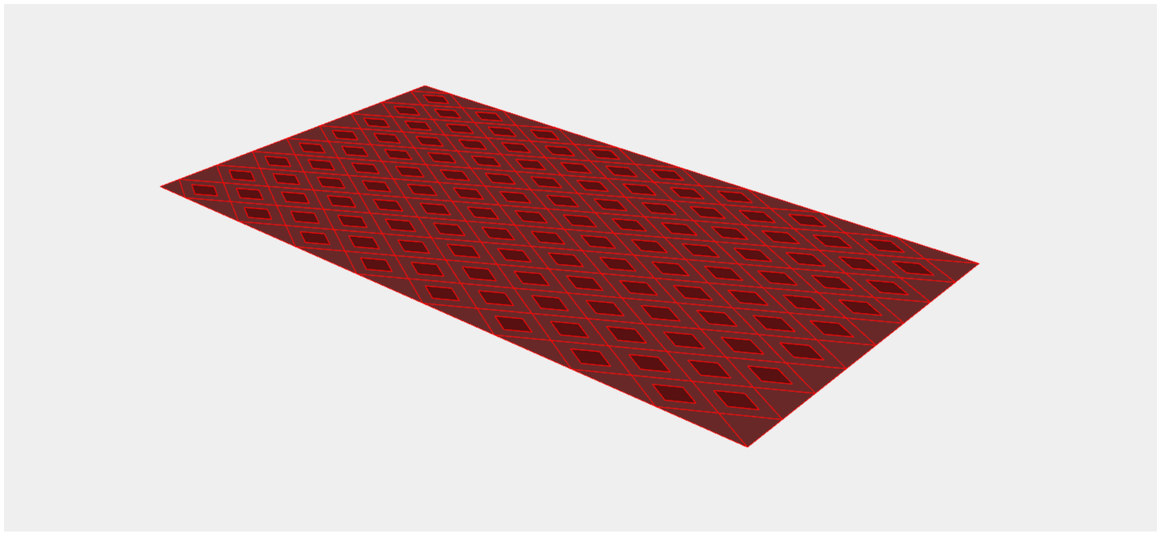


Figure 117: Diamond Paneling with scaled diamonds

Each of the scaled diamonds are moved in the positive -Z direction, and the edges of each diamond panel are extracted using a Deconstruct BREP component. The edges of the larger diamonds and the smaller translated diamonds are lofted together, capped, and then joined back together. This process is shown in Figure 118, along with an additional attractor system shown in the pink box. This attractor system works in a very similar way to the one described previously, changing the translation distance of each moved, scaled, diamond panel. The reader can review the details of this whole system in Figure 120, but the main takeaway is that a system with an entirely different design intent can be created in parallel with the other original design, using geometries defined by the original design.

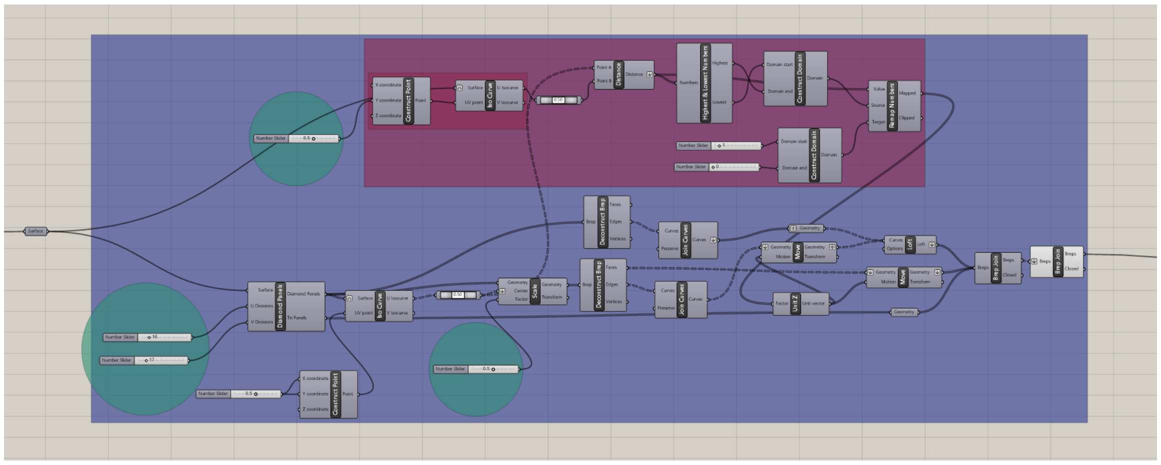


Figure 118: Entire diamond panel algorithmic system.

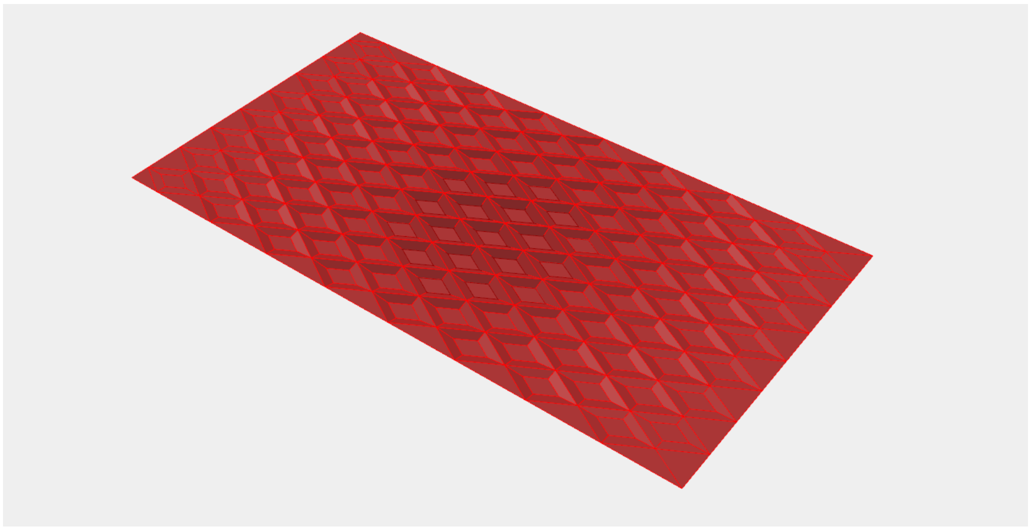


Figure 119: Final diamond paneling surface with attractor system

This diamond paneling system is then integrated back into the original system with minimal difficulty. It is placed just before the slot extruding systems are used to create the feature cuts. It is integrated using a stream filter connected to a Boolean Toggle component so the switch from one design system to another can be done with the click of a button.

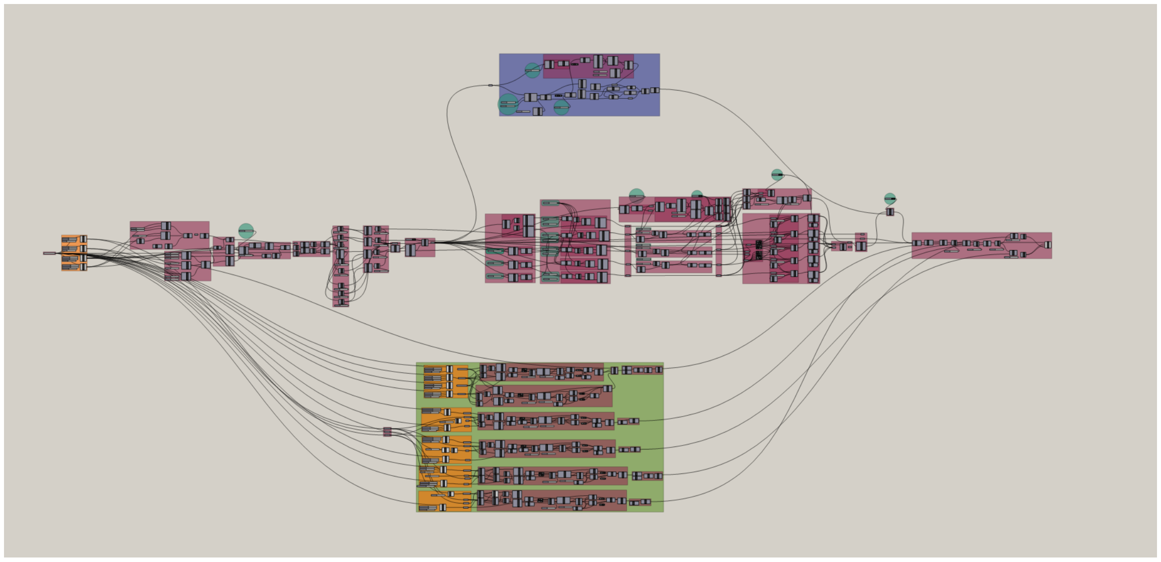


Figure 120: Diamond paneling system integration. Entire diamond paneling portion shown in the blue container box above original algorithm.

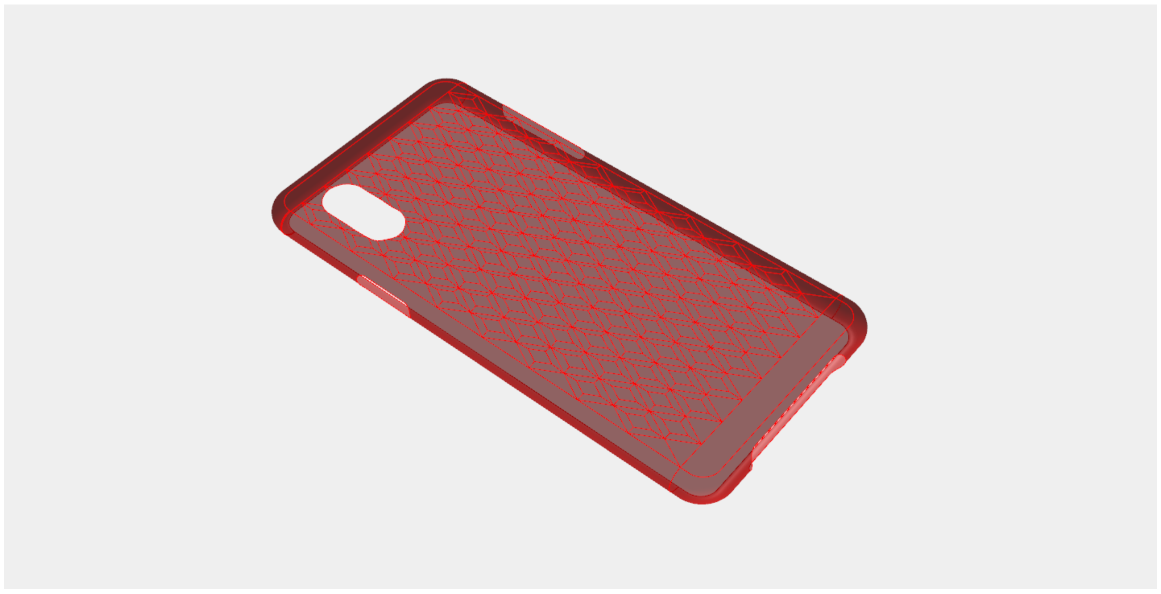
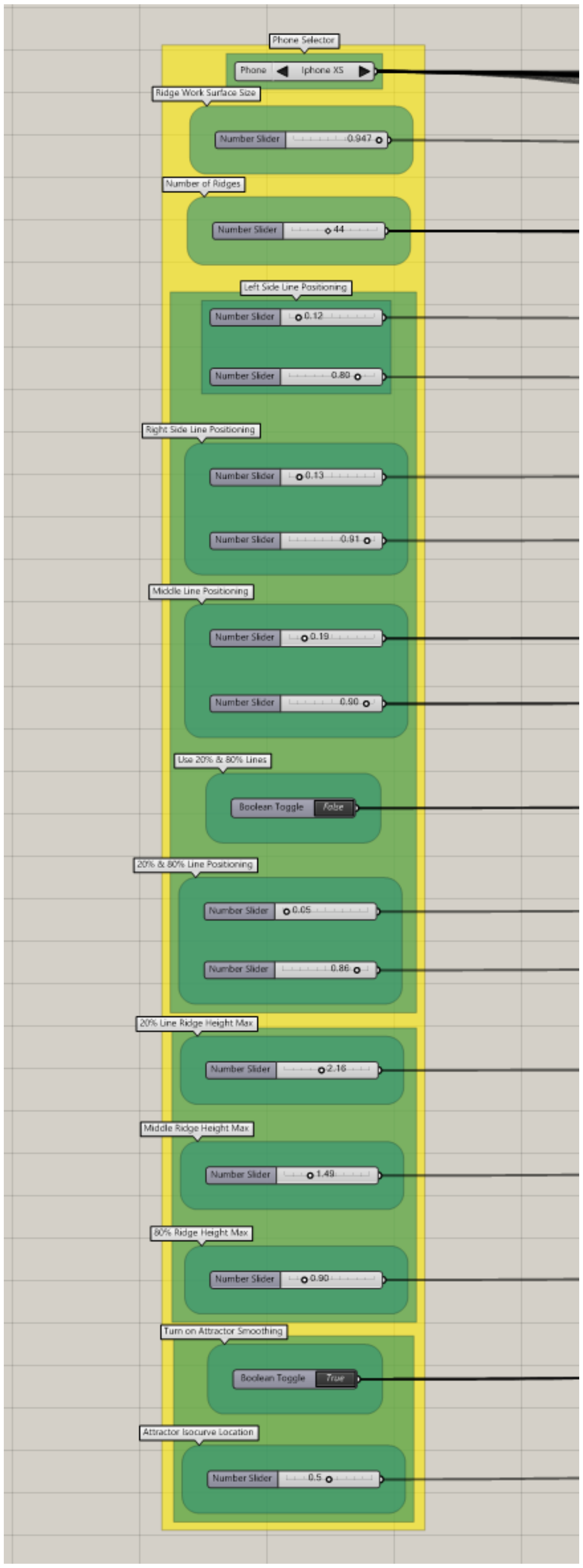


Figure 121: Completed diamond paneling option for phone case

With the final algorithmic system complete, the only step left is to organize it so that it is easy to understand and use if time has passed or if another person needs to use the system. This involves straightening lines of components, organizing components in to groups, and naming groups with their functions. The variable components which define the shape and style of the phone case were also found and brought to the very front of the model so that anyone could immediately begin attempting to adjust settings, shown in Figure 122. These adjustable parameters are not the only ones included in the model but were the ones specifically chosen to be controlled by the user. Changing these variables generally does not break the system in any way.


Phone Selector
Phone ◀ Iphone XS ▶

Ridge Work Surface Size
Number Slider 0.947

Number of Ridges
Number Slider 44

Left Side Line Positioning
Number Slider 0.12
Number Slider 0.80

Right Side Line Positioning
Number Slider 0.13
Number Slider 0.91

Middle Line Positioning
Number Slider 0.19
Number Slider 0.90

Use 20% & 80% Lines
Boolean Toggle False

20% & 80% Line Positioning
Number Slider 0.05
Number Slider 0.86

20% Line Ridge Height Max
Number Slider 2.16

Middle Ridge Height Max
Number Slider 1.49

80% Ridge Height Max
Number Slider 0.90

Turn on Attractor Smoothing
Boolean Toggle True

Attractor Isocurve Location
Number Slider 0.5

Figure 122: Adjustable settings for algorithmic phone case

After testing different options and combinations of variables in this system, the overall construction seems fairly robust. In some variations, the surface directions may flip, causing problems further down the line resulting in a broken model, but usually slightly adjusting the size of any number of dimensions can fix this without deviating far from the design intent. The following Figures will show the different variation possibilities for certain parameters with everything else remaining constant. Firstly, the algorithm allows the user to choose which phone they will be using and adjusts the model accordingly.

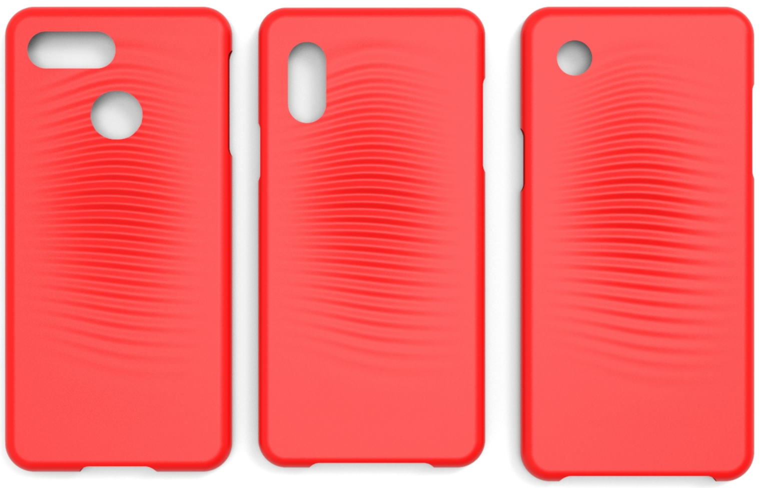


Figure 123: Different phone model choices

The algorithm then lets the user specify the size of the workspace on which the ridges will be built. Figure 124 shows how the workspace is increasingly moved toward the top of the phone, shrinking the space the ridges occupy.

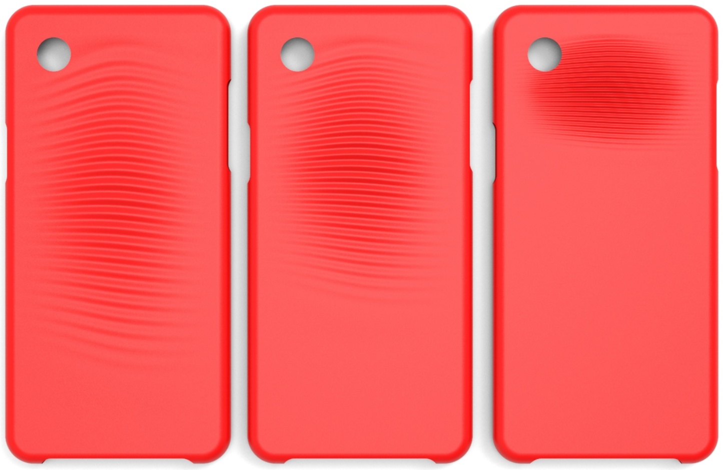


Figure 124: Variable ridge workspace sizes

The algorithm then allows the user to define the number of ridges used in the model by adjusting the number of line divisions used. This adjustment can create a great number of design variations with drastically different looks. Figure 125 shows how using a few ridges creates a flowing, subtle surface, where using many ridges can create a rougher, sharper surface.

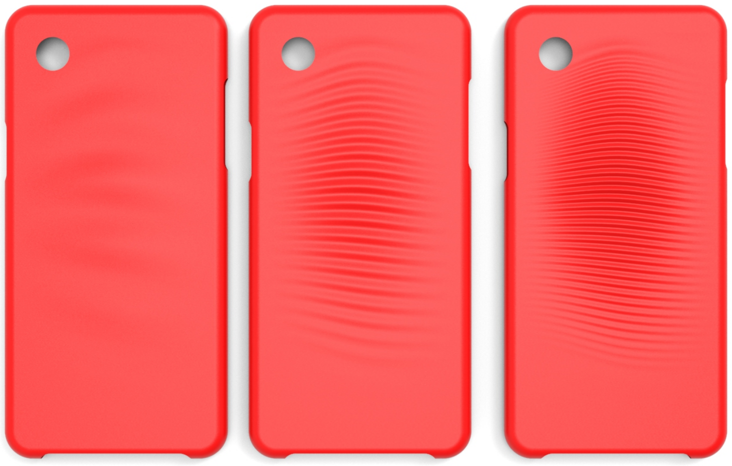


Figure 125: Varying number of ridges

The locations and size of the lines used along the edges and middle of the surface that define the control points for the splines which the ridges follow can be adjusted as well. The lines can be compressed or expanded and moved up and down the sides of the phone case. Figure 126 shows different variations of lines and how it effects the ridges. The first model on the left has a compressed left line, which is lower than an expanded right line. The middle model has an expanded left line which is higher than a compressed right line. The right model has both compressed left and right lines, but an expanded middle line. This is an area of great variability, allowing the user to greatly experiment with the design of the case.

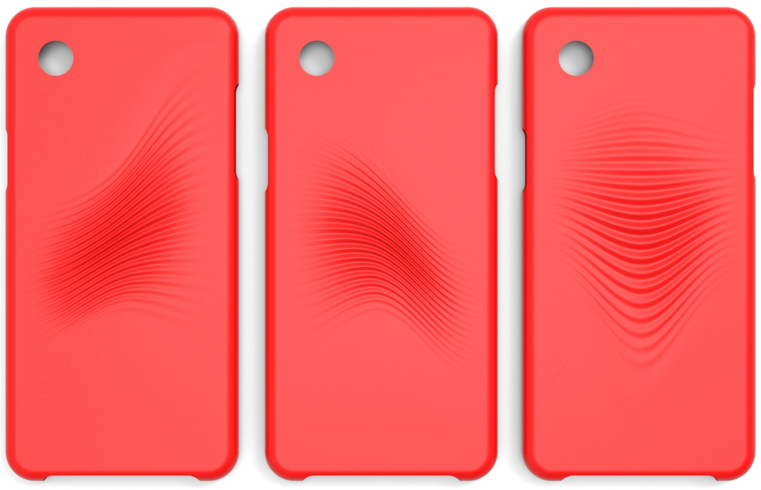


Figure 126: Variations in lines that define ridge spline control points.

Additionally, more spline control point lines can be added into the model, with two optional lines added at 20% and 80% of the width on either side of the middle line. The addition of these lines can create more chaotic looking splines but, adds a greater depth of control over the ridges.

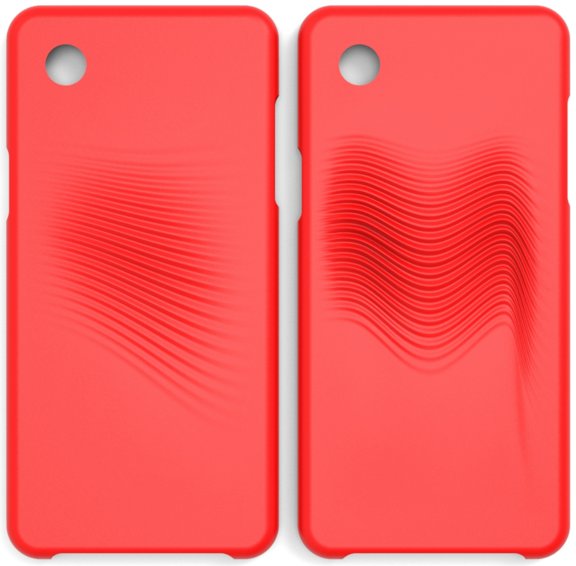


Figure 127: Left: Model with left, middle, and right control lines. Right: model with left, 20%, middle, 80% and right control spline lines.

The last major adjustment allows the user to define the maximum distance that the ridges will move off of the surface of the model. Again, choosing a small distance creates a subtler design while choosing a larger distance creates a more distinctive look.

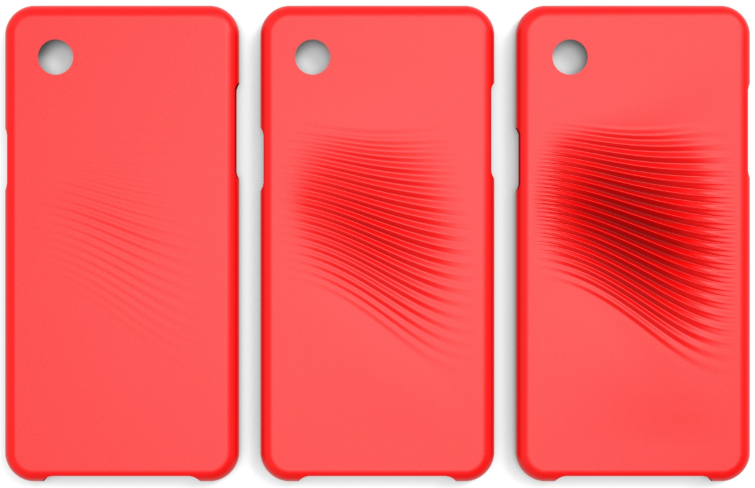


Figure 128: Variations in ridge height

All of these variations, when put together, can create a large variety of choices for the user to define for the final result of the algorithm. While the “inventive” approach of algorithmic modeling is restrictive in the sense that the designer must define the limits and capabilities of the algorithm, it can allow for a great amount of different design and iterations to be developed, as evidenced in Figure 129.

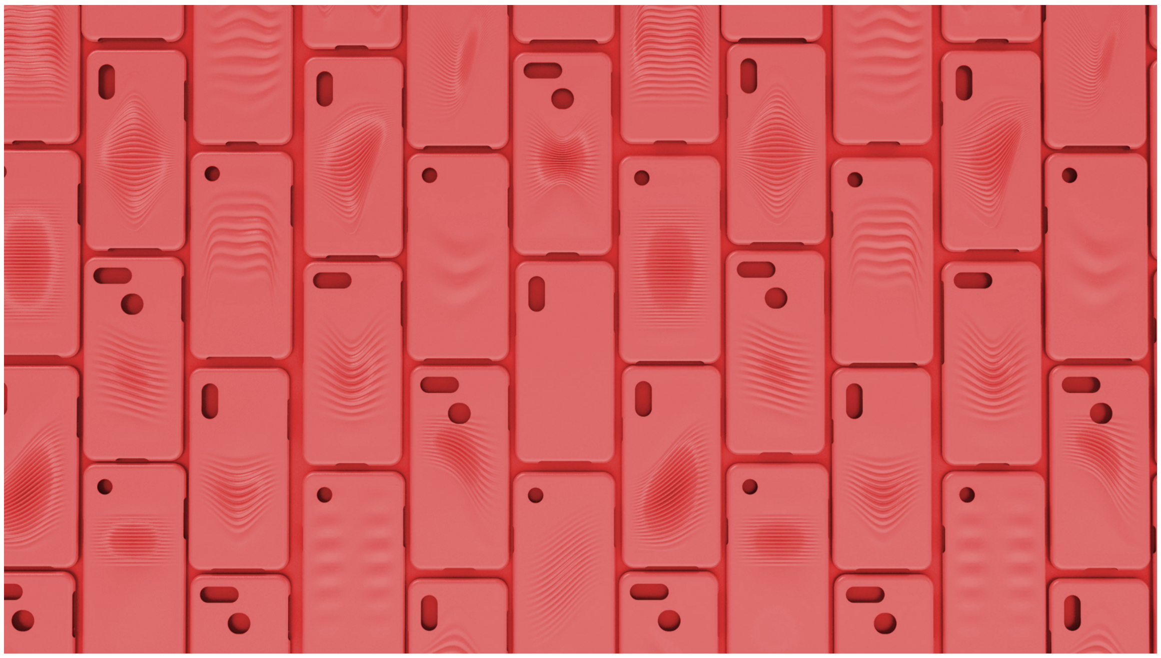


Figure 129: Many different design iterations from the same algorithmic system

It has been shown that the building process of the algorithmic system, while it does have some minor problems, was a generally straightforward process once the abstraction was developed. The success of this model does not guarantee that the algorithm was the most efficient method of creating this geometry, but it does accurately follow the abstraction process and original design intention. The entirety of the Grasshopper model can be found at: https://drive.google.com/drive/folders/1_iYBf6XisuOJ3Yhe0WXgLgz12W5Bn9bP?usp=sharing

Application Model Integration

Now that the algorithmic design is completed, it must be extracted from the system. By using Grasshopper as the chosen VPL modeler, this is a fairly simple process.

Any component containing geometry can be selected and “baked” directly into the Rhino environment. By doing this with the last component of the phone case algorithm, an open polysurface is created in Rhino, shown in Figure 130 next to the Grasshopper preview surface shown in red.

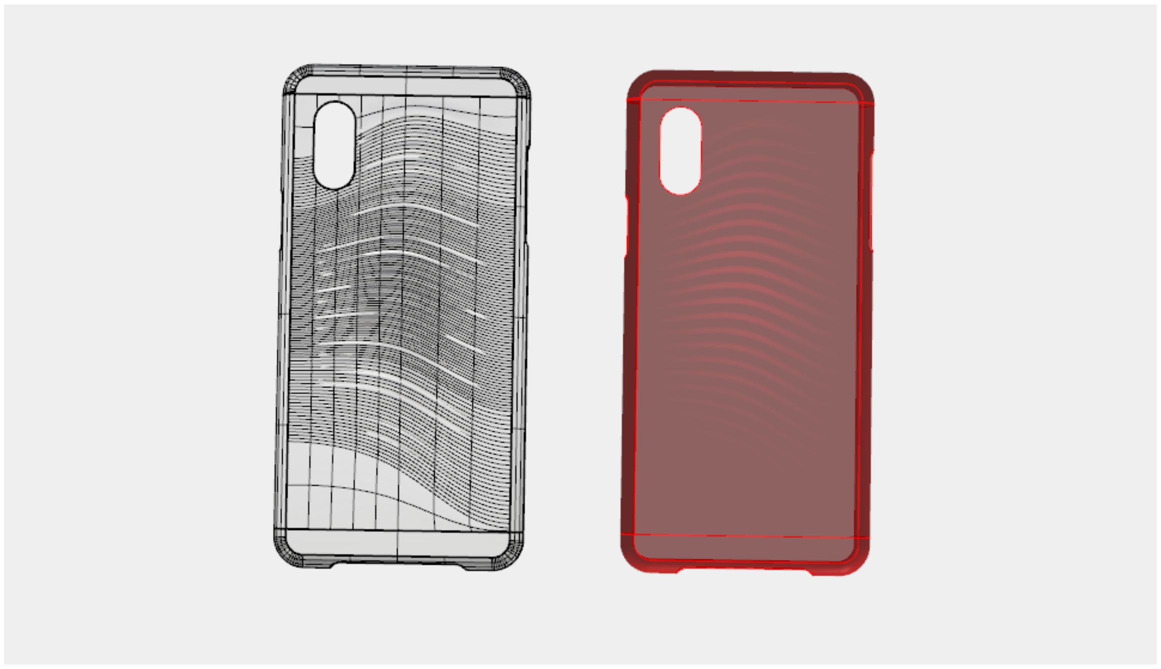


Figure 130: Baked geometry in Rhino environment.

This polysurface has no thickness, so standard Rhino tools must be used to fix this. Offset Surface is used to thicken the surface a defined amount. Interestingly, an outward offset tends to remove particular surfaces from the geometry, while an interior offset works consistently; this means the original dimensions of the phone case may need to be adjusted to allow for the interior offset size. Regardless, the phone case is now a complete, closed, solid body. This means it can be exported to any number of programs. In order to further adjust this particular model, the solid is exported as a .STEP file and imported into Autodesk Fusion 360.

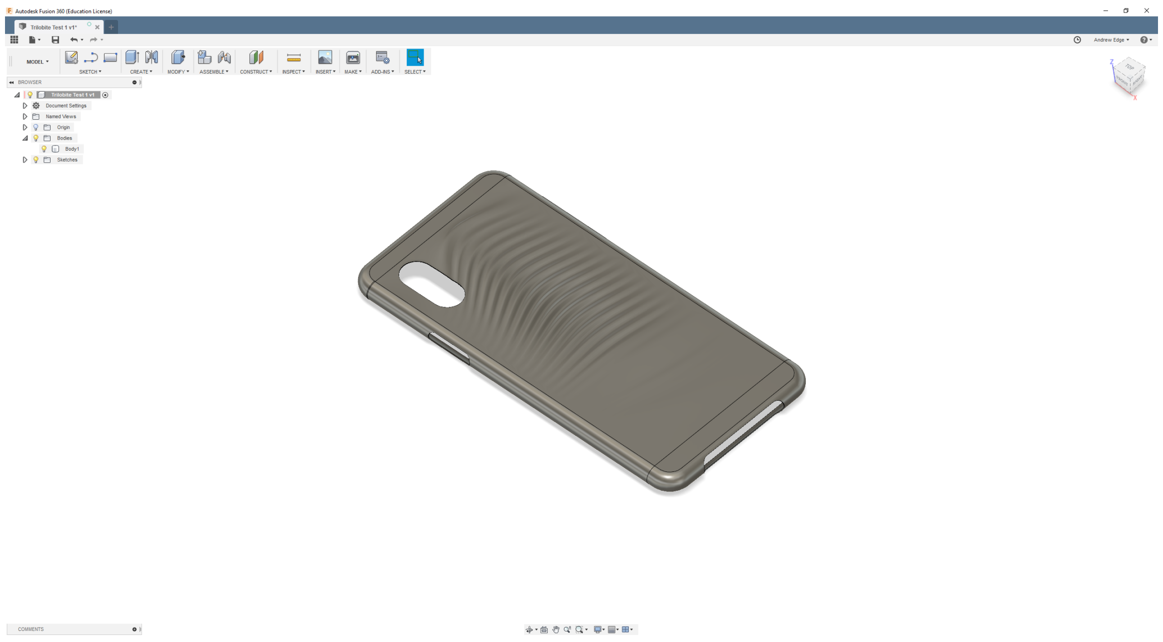


Figure 131: Phone case model in Fusion 360

The solid model can have any number of adjustments made to it, including fillets, cuts, joined extrusions etc. Figures 132 and 133 show fillets being added to the model and details like brand names being cut away, respectively.

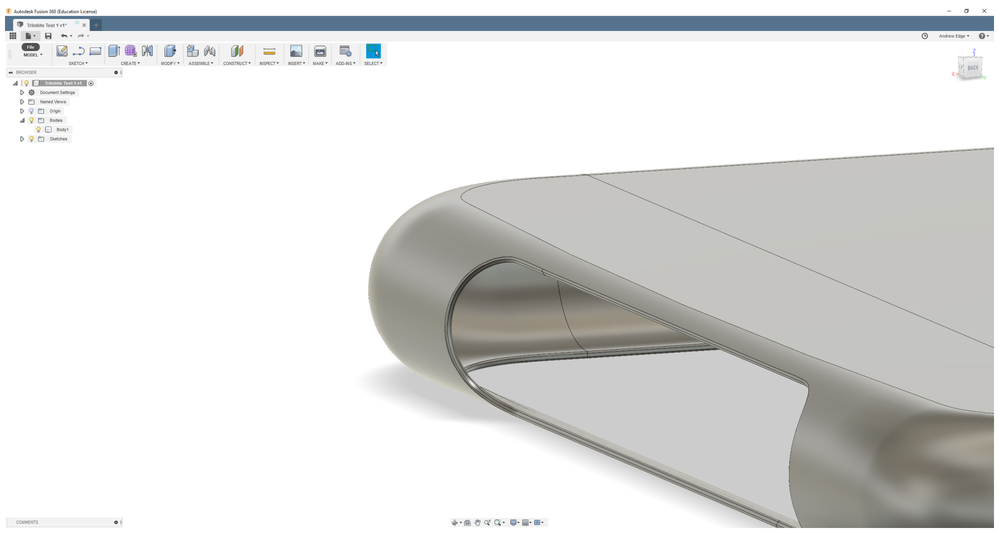


Figure 132: Fillets being added to corners in Fusion 360

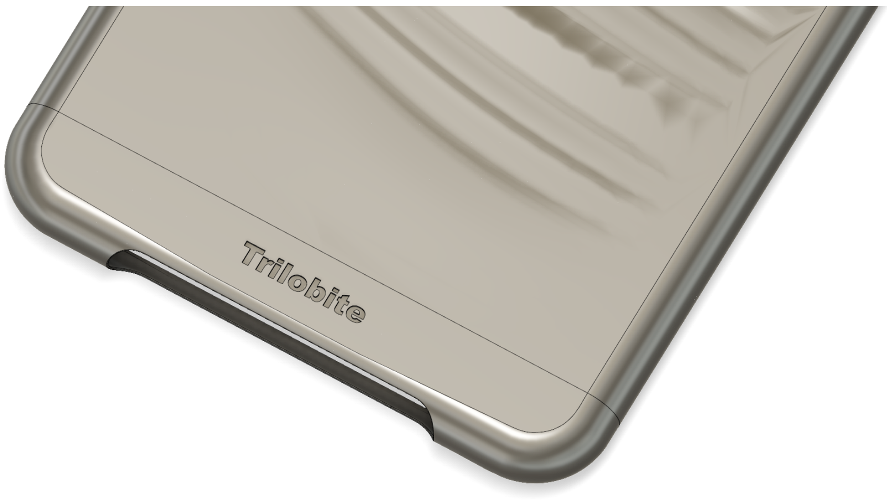


Figure 133: Brand name added from extruded cuts.

In addition to transferring the design into other modelers, it can also be exported into as a mesh for use in rendering environments. In this case, the model was exported from Fusion360 as a mesh in a .OBJ file and imported into Keyshot where it is rendered, just as any model would be. This mesh exporting process can be done directly from the baked Rhino model, as well, to save time.

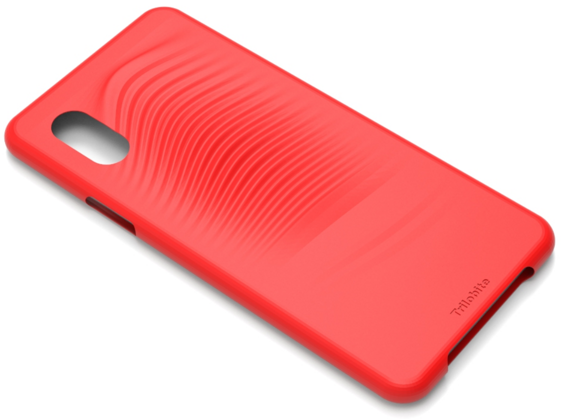
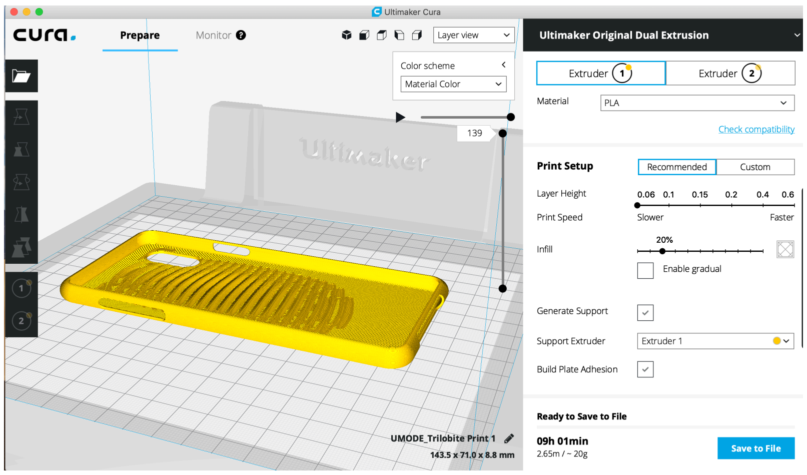


Figure 134: Keyshot rendering of exported model.

Finally, the model can also be exported as a .STL file for use in 3D printing, shown in Figure 135, where it has been imported into Ultimaker Cura, a 3D printing software.



UMODE_Trilobite Print 1
143.5 x 71.0 x 8.8 mm

Ultimaker Original Dual Extrusion

Extruder 1 | Extruder 2

Material: PLA

Check compatibility

Print Setup: Recommended | Custom

Layer Height: 0.06 0.1 0.15 0.2 0.4 0.6

Print Speed: Slower Faster

Infill: 20%

☐ Enable gradual

Generate Support: ✓

Support Extruder: Extruder 1

Build Plate Adhesion: ✓

Ready to Save to File

09h 01min
2.65m / ~ 20g

Figure 135: Model in Ultimaker Cura, preparing to be 3D printed.

Application Summary

This project worked through the entire design process from ideation to a fully developed 3D model and form for the creation of a phone case. By following the guidelines defined in this document, a concept was developed, and the design situation surrounding it was evaluated according to the considerations listed to determine its validity for algorithmic modeling. Once the design was deemed to be appropriate, an abstraction for the algorithmic process was formed for the creation of the model, through simple sketches. Now that a roadmap for the algorithmic design was created, Grasshopper was used as the chosen algorithmic modeler to define the phone case. By building this system, the desired design was algorithmically defined along with additional parameters not initially in the abstraction and the algorithm easily allows for diverging design systems to be created. The algorithm has minor issues related to surface flipping at specific parameters and some testing will be needed to determine if the hard-coded sizes of the phone dimensions are accurate, but overall, the algorithm performs as expected and can create a large variety of models that reflect the original design intent. The first output of the system, the phone case model, can easily be integrated into other modelers, renderers, and 3D printers. The second output, the algorithm itself, is organized well, provides great variability to the user, and should be able to be created in other algorithmic systems that follow its logical flow.

Chapter 6: Conclusions

Summary

This thesis was written to explore the definition of algorithmic design and its place within the industrial design process. Humans have always needed plans for how to build things, but the methods by which they create and translate their plans have continually changed. Plans or instructions on how to complete any process are algorithms and recent technologies and programs have emerged which allow algorithms to define geometries easier than ever before and to use those computer-defined geometries as direct inputs into the manufacturing process. The algorithmic modeler and the algorithmic modeling process are somewhat foreign to the typical industrial design education and skillset and are processes which consume far more time in their use than traditional 3D modelers.

Given these difficulties, this thesis provides guidelines for the practical use of algorithmic modeling as part of the typical industrial design process. First, a concept must be developed through traditional industrial design methodologies: research, sketching, iteration etc. Once the final concept is selected, it must be evaluated for its viability as a candidate for algorithmic modeling, using several considerations of the design and the design situation it is placed in. If it is decided that algorithmic modeling is appropriate, the designer must form an abstraction of the algorithm to be used in the modeling software, and then the chosen algorithmic modeler must be used to define the model before outputting a final geometry for use in other softwares and systems.

This thesis also provides explanations of common algorithmic terms which may be helpful when working through the abstraction process and may provide insight into how different algorithmic modeling components operate. Also provided are examples of several commonly used algorithmic techniques which may serve as starting points for future designs or merely to show steps which can be used in other modelers.

This thesis demonstrates that algorithmic design can be efficiently used in the industrial design process and provides a detailed look at an industrial design case study of the creation of a smart phone case design. It was shown that the guidelines provided effectively determine whether a project is suitable for algorithmic modeling and an in-depth look at a Grasshopper modeling process was performed, resulting in the creation of a successful phone case modeling algorithmic system.

Future Research

This thesis focuses on the "inventive" side of algorithmic modeling, as compared to the "discovery" side. Future additions to this work could determine what conditions and methods are required for effective use of truly generative design systems and whether or not these systems are effective at outputting designs that match the designer's intention. Often generative design systems have unpredictable or very organic looking outputs, as is their nature, but an evaluation of their output as a product or merely as a learning tool for other designs could be beneficial to their use in industrial design. Additionally, the nature of algorithms allows for expansion into design territory that isn't 3D modeling based, so work could be done in the fields of graphic design, AI, organization and more.

References

1963 | Timeline of Computer History. (n.d.). Retrieved from https://www.computerhistory.org/timeline/1963/

Agkathidis, A. (2016). *Generative design: Form-finding techniques in architecture*. Laurence King Publishing.

Aiello, C. (2014). *Digital & parametric architecture*. EVolo.

Argan, G. C., & Robb, N. A. (1946). The Architecture of Brunelleschi and the Origins of Perspective Theory in the Fifteenth Century. *Journal of the Warburg and Courtauld Institutes, 9*, 96. doi:10.2307/750311

Bézier, P. (1974). Mathematical And Practical Possibilities Of Unisurf. *Computer Aided Geometric Design,* 127-152. doi:10.1016/b978-0-12-079050-0.50012-6

Cadazz. (n.d.). CAD software - history of CAD CAM. Retrieved from http://www.cadazz.com/cad-software-history-1990-1994.htm

Carlbom, I., & Paciorek, J. (1978, 12). Planar Geometric Projections and Viewing Transformations. *ACM Computing Surveys, 10*(4), 465-502. doi:10.1145/356744.356750

Colburn, T., & Shute, G. (2007, 06). Abstraction in Computer Science. *Minds and Machines, 17*(2), 169-184. doi:10.1007/s11023-007-9061-7

Dictionary by Merriam-Webster: America's most-trusted online dictionary. (n.d.). Retrieved from https://www.merriam-webster.com/

Dunn, N. (2012). *Digital fabrication in architecture*. L. King.

The Dynamo Primer. (n.d.). Retrieved from http://dynamoprimer.com/en/

Grasshopper. (2018). Retrieved from https://www.grasshopper3d.com/

Grasshopper Component Reference. (n.d.). Retrieved from https://rhino.github.io/#System

Haselberger, L. (1985, 12). The Construction Plans for the Temple of Apollo at Didyma. *Scientific American, 253*(6), 126-132. doi:10.1038/scientificamerican1285-126

Holcomb, M., & Bessette, L. (2009). *Pen and parchment: Drawing in the Middle Ages ;*. Metropolitan Museum of Art.

Isaacson, W. (2018). *Leonardo Da Vinci*. Simon & Schuster Paperbacks.

Levy, S. (2017, September 06). This Computer Language Is Feeding Hacker Values into Young Minds. Retrieved from https://www.wired.com/2017/05/this-computer-language-is-feeding-hacker-values-into-young-minds/

Menges, A., & Ahlquist, S. (2011). *Computational design thinking*. J. Wiley & Sons.

Monge, G., Monge, G., & Monge, G. (1798). *Géométrie descriptive*. Baudouin.

Narayan, K. (2013). *Computer aided design and manufacturing*. PHI Learning.

Open source graphical programming for design. (n.d.). Retrieved from http://dynamobim.org/

Sakamoto, T. (2009). *From control to design: Parametric/algorithmic architecture*. Actar.

System, I. N. (n.d.). Nervous System. Retrieved from https://n-e-r-v-o-u-s.com/

Tedeschi, A., Andreani, S., & Wirz, F. (2016). *AAD_Algorithms-Aided Design: Parametric strategies using Grasshopper*. Le Penseur Publisher.

Terzidis, K. (2012). *Expressive form: A conceptual approach to computational design*. Routledge.